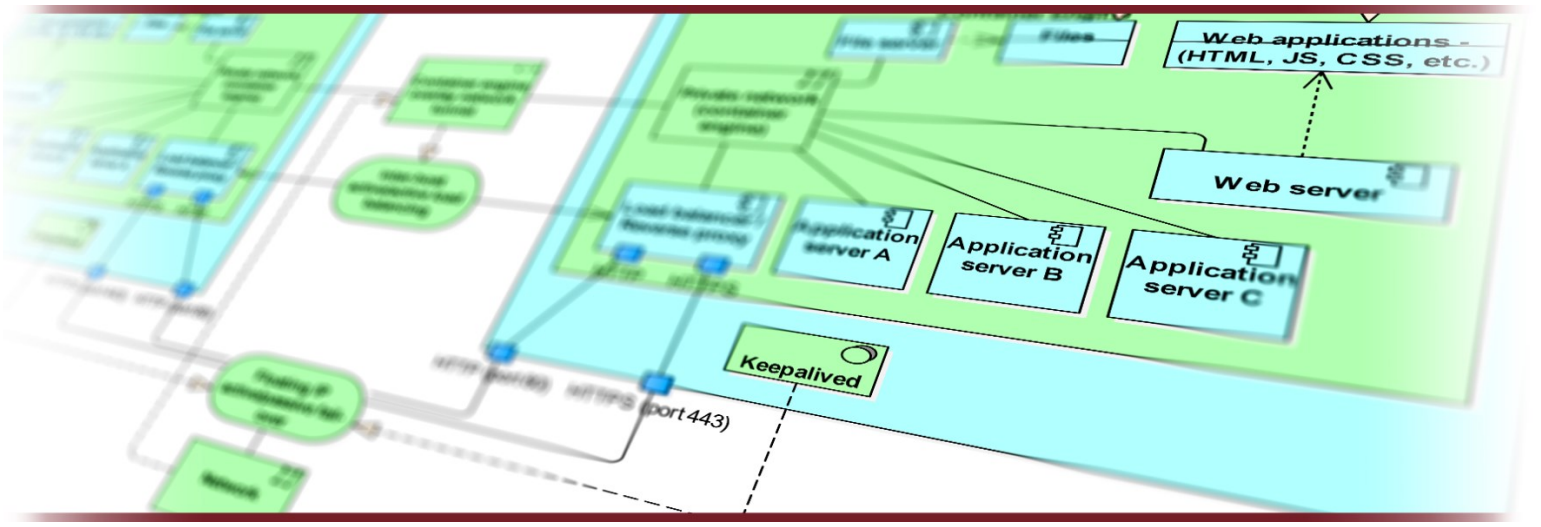
An architectural design for LAN-based web applications in a military mission- and safety-critical context



Master thesis

Non-public version

S.F. van Langen
University of Twente
Master Business Information Technology
Faculty of Electrical Engineering, Mathematics and Computer Science

Page intentionally left blank

An architectural design for LAN-based web applications in a military mission- and safety-critical context

Master Thesis
21st of October 2016

Author

Ferdi van Langen
Study programme — Master – Business Information Technology
Track — IT management
Faculty — *Electrical Engineering, Mathematics and Computer Science*

Graduation Committee

Dr. M.E. Iacob — **First supervisor**
Faculty — Behavioural Management and Social Sciences
Department — *Industrial Engineering and Business Information Systems*

UNIVERSITEIT TWENTE.

Dr. L. Ferreira Pires — **Second supervisor**
Faculty — Electrical Engineering, Mathematics & Computer Science
Department — *Services, Cyber-security and Safety*

UNIVERSITEIT TWENTE.

H.W.K. Boenink — **External supervisor**
Department — Thales Nederland – Naval Applications Engineering
Function — *Infrastructure Architect*

THALES

Preface

Dear reader,

This master thesis is the result of my research at Thales, and with it my time as a student comes to an end. That road has taken a bit longer than originally planned when I started my first year at the University of Twente several years ago, as there have been some distracting, yet incredibly enriching, extracurricular activities along the way.

During the course of my final research project several people have had positive influences on the realization of this thesis. So, first, I would like to thank my university supervisors, Maria Iacob and Luís Ferreira Pires, for their patience with the progress of my work and their effort and flexibility in helping me navigate my encounters with the university bureaucracy. Secondly, I would like to thank my external supervisor at Thales, Willy Boenink, for giving me a lot of helpful feedback regarding the workings of the TACTICOS system, and for giving me the time and opportunities to learn a lot more about all aspects of web technology. I would like to thank the Thales employees that contributed to my research for their insights and feedback. I would like to thank my fellow students at our department in Thales for the engaging conversations during the lunchbreaks, and I would like to thank the Thales employees of the 2C.60 spaces for making the workdays more pleasant. And, finally, I would like to thank my parents for their continuous support during the full length of my study.

Due to the confidential nature of the case study used in this research, certain results and descriptions could not appear in the public version of this thesis. Any information that came up in this research regarding the inner workings of the TACTICOS Combat Management System I've kept out of the public version of the thesis, and replaced it with a notification that the paragraph or section in question was redacted by me. I've tried to keep an as clear as possible separation between paragraphs dealing with confidential results and paragraphs discussing my methodology or containing public information, so you can still follow as much as possible about what I have done during my research.

I hope this thesis gives you, the reader, an opportunity to come into contact with some technology and related principles you've not yet encountered before.

Kind regards,
Ferdi van Langen.

Management Summary

The requirements of software systems can change over time, due to new business opportunities, new threats, new technologies and other factors. In this case a study was done into how web technology can be incorporated into (military) mission- and safety-critical systems, like naval Combat Management Systems (CMS). These systems can differ from regular enterprise systems as failures of the system can result in the loss of life, which leads to different considerations during their development. To explore the demands on a (military) mission- and safety-critical system a case study was done at Thales Netherlands B.V. regarding their TACTICOS Combat Management System and how web technology could be integrated into it.

This research started with an analysis of what the strengths and weaknesses of the current architecture of the TACTICOS system were, and what opportunities and threats (modern) web technology could introduce. The current architecture of the TACTICOS system was also modelled in architectural diagrams scoped to a few representative use cases. A literature search was done into the architectural possibilities of web technology, and guidelines on how to integrate web technology were given. Those guidelines were applied to the current architecture of the case study, to create a new architecture for the specified use cases. This was then evaluated by company experts. Another part of the evaluation was making several (smaller) prototypes for familiarization with web technology and testing the feasibility of certain concepts surrounding web technology.

There were found to be limitations in the current architecture of the TACTICOS Combat Management System that could be solved with the integration of web technology into the system's future architecture. Next to this, several new opportunities could be unlocked in the system by web technology. Web technology was found to be beneficial for (military) mission- and safety-critical systems, due to the uptime maximizing focus in web technology based projects, and also its modular nature.

Due to this research being a master thesis the scope had to stay relatively small and it cannot be guaranteed that the results from this research can be extrapolated to a complete system. Also, there is little information to be found about competitor's products, and thus the validity of this research's results as a generic solution could not be ascertained. While there is plenty of literature on the performance and reliability of web technology facing a large amount of short-lived low-workload connections, the same cannot be said for situations with a small amount of long-lived and high-workload connections, as is the case with these kinds of military mission- and safety-critical systems, which could be a rewarding area for future scientific efforts to be directed to.

Based on the results of the case study the recommendation would be for Thales to (immediately) start looking into the performance of web technology under a realistic load in a realistic test set-up. This can give insight into which parts of the TACTICOS ecosystem can be feasibly replaced by web applications, and more importantly: which parts cannot. Certain parts of the TACTICOS system have less demands in regards to performance, latency and reliability than others, and thus can be more eligible for a move to web technology, realistic load testing can provide information necessary of where to place the bar. In the short-term the recommendation would be to start a pilot project to increase the knowledge about and find the limitations of web technology. This pilot project should be small in scope, but still representative of a typical segment of the TACTICOS system. With the information from these projects a roadmap can be drawn up, for the long term, of how web technology can be safely rolled out in the TACTICOS system.

Page intentionally left blank

Table of Contents

Preface ..... ***I***

Management Summary ..... ***II***

Table of Contents ..... ***IV***

Chapter 1 - Introduction ***1***

1.1 Motivation *1*
1.2 Background *2*
1.3 Research Proposal *3*
1.4 Research Methodology *5*
1.5 Thesis Structure *6*

Chapter 2 - Problem Analysis ***8***

2.1 Case study context *8*
2.2 Approach *8*
2.3 Interviews with experts *9*
2.3.1 Strengths of the current architecture 11
2.3.2 Weaknesses of the current architecture 11
2.3.3 Opportunities for a new architecture 12
2.3.4 Threats to a new architecture 13
2.4 Requirements for the new architecture *14*

Chapter 3 - Modelling The "As-Is" Architecture ***16***

3.1 Use case 1: 3D search radar *24*
3.2 Use case 2: Video server *25*
3.3 Use case 3: training simulations *28*

Chapter 4 - Architectural Styles ***31***

4.1 Literature Review - Search & Selection Strategy *31*
4.2 Architectural Styles *32*
4.3 Guidelines for a new architecture *34*

Chapter 5 - Design ***38***

5.1 Use case 1: 3D search radar *38*
5.2 Use case 2: Video server *41*
5.3 Use case 3: Training simulations *43*
5.4 Web applications servers – internal architecture *48*

Chapter 6 - Evaluation ***52***

6.1 Design evaluation *52*
6.2 Prototype evaluation *54*
6.2.1 Sensor simulator 54
6.2.2 Publish/Subscribe service registry 56
6.2.3 OpenSplice Data Distribution Service 58
6.2.4 Custom browser with local content 60
6.2.5 Containerized microservices 62
6.2.6 HTTPS/TLS security 64

Chapter 7 - Conclusions & Recommendations ***69***

7.1 Conclusions *69*
7.1.1 Research Questions 69
7.1.2 Main Conclusion 70
7.1.3 Limitations 70

7.2 Recommendations 70
7.2.1 Immediately relevant recommendations for Thales 70
7.2.2 Recommendations relevant in the short term for Thales 71
7.2.3 Recommendations relevant in the long-term for Thales 71
7.2.4 Future work 71
7.2.5 Business practice 71

References 72

List Of Figures And Tables 78

Index 80

Appendix A: Search terms used in the literature review 81

Appendix B: Architectural styles found in the literature search 83

Appendix C: Architecture evaluation questionnaire 85

Chapter 1 - Introduction

1.1 Motivation

Maintenance accounts for more than half of the costs for a software system's total costs during its lifetime, and it has been rising over the years (Østerlie & Wang, 2006). The systems being maintained have become more complex, and because of this, more developers are needed to maintain the systems, performing upgrades and fixing bugs. Maintainability is the measure of how easy it is to correct, improve or adapt a system. The maintainability of software systems is one of the most important concerns for developers (Coleman, Ash, Lowther, & Oman, 1994). The complexity of a system causes the interactions between the components to become one of the major factors for the reliability of a system, and thus these interactions should be taken into account when looking at its maintainability (Thomas, 1986). A model that describes different interactions of the system can provide information about its maintainability, and thus guide the maintenance efforts (Coleman, Ash, Lowther, & Oman, 1994). Measuring maintainability can be helpful to assess current systems and support decisions in the design of a new system (Østerlie & Wang, 2006).

Businesses have an increasing need for flexibility and agility to adapt to changes, which can be provided by an architectural design supporting that flexibility and agility (Aerts, Goossenaerts, Hammer, & Wortmann, 2004). When the flexibility of a system is not adequate this can lead to high costs, service disruptions, lost opportunities and value loss (Engel & Browning, 2008). Flexibility of a system can be increased by preparing the architecture for later modifications that will adapt the system to unpredicted circumstances, so that due to this flexibility the architecture becomes more maintainable (Mari & Eila, 2003).

Typical reasons to upgrade components of a system are either corrective, wherein bugs are fixed; perfective, to improve the performance and functionality of a system; and adaptive, to adjust the system to a changed environment (Paspallis & Papadopoulos, 2007). When technology changes with a fast pace modular upgradeability is preferred. A modular architecture can structure components in such a way that upgrades can be done in incremental small steps by upgrading a single (relatively decoupled) module each time instead of upgrading a whole monolithic system. This way costs can be minimized by only doing work on the components that need upgrading instead of working on an integral system (Kamrad, Schmidt, & Ulku, 2013). Another advantage of a modular architecture is that new modules and interfaces can be added easily (Engel & Browning, 2008). After modifying a system one has to validate whether the modifications on the system are in working order: the ease with which this can be done is referred to as the testability of a system (Mari & Eila, 2003). Replacing components should preferably be done with minimal downtime of the system (Paspallis & Papadopoulos, 2007). Upgrades that can be done without needing to shut down the (whole) system, using an architecture that supports dynamic upgrades, can thus become more cost-effective and open up more time-windows in which upgrades can be done to a high-availability system.

Desktop computers and workstations lack mobility, and because of that there are opportunities for mobile devices to enrich the user experience previously provided by those stationary computers (Abolfazli, Sanaei, Gani, Xia, & Yang, 2014). The mobility that devices provide can be beneficial for productivity, as it increases agility in where you handle business processes, lowers costs, improves business processes and services and can support decision-making at more places. With a mobile device, information can be accessed any place, anytime (Pérez, Cabrerizo, & Herrera-Viedma, 2010), and these devices have the advantage of portability, location awareness and accessibility (Nayebi, Desharnais, & Abran, 2012). Ubiquitous computing is considered a revolution in IT, since it is about providing (information) services adapted to the user and to the user's context. Some ubiquitous computing approaches only reproduce the functionality of traditional systems, whereas more innovative approaches protect the user from poorly relevant data by being personalized and context-

dependent (Spaccapietra, Al-Jadir, & Yu, 2005). Rich Mobile Applications (RMAs) provide "high functionality, seamless ubiquity, context-awareness, and immersive interaction" on location (Abolfazli, Sanaei, Gani, Xia, & Yang, 2014, p. 23), since they have their Rich Internet Application (RIA) functionality that is then enriched by several benefits gained from the use of mobile devices to interact with these applications. RMAs can enhance the quality of work done on mobile devices (Abolfazli, Sanaei, Gani, Xia, & Yang, 2014). Smartphones and tablets give additional interactivity options, like multi-touch & voice commands, enhancing the user experience over traditional human machine interaction (Wang & Canedo, 2014). Web technology can lower the threshold for using mobile devices, and can thus provide better productivity.

As there are more and more innovative solutions in modern technologies, and the duration of a product's lifecycle is becoming more and more volatile, the same speed of innovation is starting to become needed in military mission- & safety-critical systems. This is further compounder by the need to be more agile in dealing with threats to the security of the system.

1.2 Background

The architectural demands for computer systems in naval ships and specifically command & control systems (C2 systems) differ from those in ordinary architectures due to the irregular demands placed on such systems, such as differences due to speed of action, mobility and security considerations confounded by, for example, external conditions that pose a threat of life (Pilarski, 2014) as well as "fast-moving forces, rapidly-changing situations and an abundance of data" (Bennett, 2014, p. 341). The trend is towards more decentralized Command & control systems, without a single person having the complete tactical overview of the situation. Instead, tasks are spread out among operators, each responsible for a small subset of tasks, creating a 'shared situational awareness' (Gorman, Cooke, & Winner, 2006). Military systems have human factors as performance degradation and cognitive complexity of issues and command style involved, due to the distributed nature of military teams and the stress imposed on decision makers (Tolk, 2012). The addition of mobile devices to access such C2 systems can heighten team situational awareness by sharing adaptive and timely information (Gorman, Cooke, & Winner, 2006) in situations and locations where access to the stationary operator consoles is not available.

A system can be tagged as a safety- or mission-critical system based on the consequences of a mishap: a system is regarded as a mission-critical system if the consequences of a severe mishap result in the failure to complete an organizations operation, be it a business operation or a military mission. In a safety-critical system, a serious mishap results in physical harm or even the loss of life. Systems can also be considered as a combination of these definitions to varying degrees. Mission- and safety-critical systems need to be developed thoughtfully and carefully (Fowler, 2004). The development of safety-critical systems is different from regular enterprise systems, as avoiding injuries or loss of life takes precedence over other factors when designing such a system (Bowen J. P., 2000). The dependability of a system can be increased by a combination of several approaches to handle failures. ((Laprie, 1989) as cited by (Bowen & Stavridou, 1993))

When talking about architectures in software systems one can make a distinction between three levels:

- The tangible physical systems in a real deployment.
- The abstract arrangement of relations between systems in a real or proposed solution
- The models that represent these arrangements of relations between systems

In this thesis, when talking about architectures, the abstract arrangement of relations between the systems is meant. The third level will be referred to as architectural models. Architectural models are a representation of the way in which a system is composed of its components. With an architecture one can show how different components connect to and use each other, as well as why the choices behind the architecture were made.

The modelling 'language' in which an architecture is represented is called an Architecture Description Language (ADL). Different ADL's provide different (visual) notations to represent process, data, and connecting elements. Archimate is an example of an ADL for broad use. While an architecture is a concrete arrangement of elements for a specific system, an architectural style is the abstraction of the characteristics that a group of architectures (with a specific purpose) have in common. An architectural style is the blueprint with which to design new architectures. An architecture can also be used as an architectural style (as a blueprint) for another architecture. Enterprise Architecture Frameworks (EAF's) are a methodical way to organize information about an architecture. Because of the structured nature of EAF's one can use them to compare different architectures or styles. Two architectures can be compared with a gap analysis. Based on that a migration plan (or transition plan) can be made, describing the strategy to move from one architecture to another. (Britton & Bye, 2004) (Perry & Wolf, 1992) (ISO/IEC/IEEE, 2011) (Alghamdi, 2009) (Rouhani, Mahrin, Nikpay, & Rouhani, 2014) (Armour & Kaisler, 2001)

Next to functional requirements that specify the actions of software systems, non-functional requirements specify constraints and quality attributes for these systems (EAGLES Evaluation Working Group, 1996). The quality attributes can be defined in a product quality model, like with the ISO/IEC 25010 standard (as cited by (Wagner, 2013))[1]. Risks are uncertain events or conditions that can affect quality characteristics (Project Management Institute, 2004). As it is uncertain whether a risk will occur, it is also uncertain whether its consequences will happen. Because of this uncertainty, the severity of a risk can be defined as a product of its probability and its impact (Zur Muehlen & Ho, 2005). A specific kind of risk, where the consequences of the risk depend on whether there is a specific vulnerability in the system that can be impacted, is called a 'threat'. Threats can be either deliberate or accidental (Im & Baskerville, 2005).

1.3 Research Proposal

The motivation in section 1.1 leads to the following research goal:

"Design a new architecture and an architectural migration strategy for a combat management system, to exploit the benefits of web technology."

To achieve this goal several questions were asked, which were designed to cover the steps of the Design Science Research Methodology (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007), to systematically achieve the research goal. Each question relates to one or more steps of the DSRM.

As concrete representation of the kind of combat management systems discussed in this thesis a specific case study was used: The TACTICOS combat management system as designed and developed by Thales Netherlands. Thales has a need to look into the use of (LAN-based) web applications and services to better prepare their military mission- and safety-critical systems for future needs relating to upgrade- and maintainability, expansion to mobile devices, and easier addition of new data-sources & applications. In their existing systems with a large codebase of legacy functionality it is not feasible to make a single all-encompassing transition to a new architecture involving web applications due to the amount of 'legacy' systems, but a (partial) transition involving several increments should be structured. Their combat management system is used to model a baseline architecture, what future possibilities provided by web technology are expected, and the proposed guidelines for a new architecture are applied to the TACTICOS architecture.

[1] ISO/IEC 25010 standard - product quality model: functional suitability; reliability; performance efficiency; usability; maintainability; security; compatibility; portability (as cited by (Wagner, 2013)).

	Research Questions
RQ1	*What limitations are encountered in the current architecture?*
RQ2	*What are the requirements/constraints for a new architecture in a CMS by Thales?*
RQ3	*What kind of architectural styles are suitable when using web applications and services in a CMS?*
RQ4	*What risks to the product quality could a new architecture encounter?*
RQ5	*What should a new CMS architecture look like to support web applications and services?*
RQ6	*How should the (long-term) migration path towards the new architecture be structured?*
RQ7	*Is the proposed architecture suitable for a CMS?*

Table 1: Research questions

In RQ1 the goal was to find out how the current architecture is structured and what its limitations are. In this case, not only the limitations were considered, but also possible unrelated problems that should be considered when designing a new architecture. RQ2 was used to find out what requirements/constraints would be placed on a new architecture by Thales employees, for example, regarding rules & regulations, internal best practices, available hardware, etc. RQ3 is a detailed inquiry into the different architectural styles to structure the architecture, which needed to be done before a new architecture could be designed. In RQ4 the risks that could possibly be encountered by the architecture, as well as possible weak spots in its quality aspects, were identified. These are different from standard enterprise architectures due to the specific nature of military combat management systems. RQ5 was used to propose a newly designed architecture for the TACTICOS system, taking into account the answers from question 1 through 4. Parallel to the previous question, in RQ6 a migration path towards the desired architectural changes was considered, for Thales to be able to selectively choose in the future when parts of the new architecture are to be implemented. In RQ7, the suitability of the proposed architecture and its migration path was determined against the measures of maintainability, flexibility, upgrade- and expandability, and mobile capabilities.

As time and resources for a thesis are limited, the scope has been carefully defined. To achieve this scope, this research looks only at architectural styles involving web applications and services that are relevant for the new architecture to achieve more flexibility. In addition, when looking at the risks against product quality, risks involved at ordinary enterprise architectures are relatively well known, and can be ignored to focus on risks and threats that are involved within a military and/or mission- and safety-critical context. Furthermore, for the risks against product quality, of the 8 categories in the ISO/IEC 25010 standard[2], this research mainly focuses on maintainability, reliability, and portability. The 5 remaining categories, performance efficiency, security, compatibility, functional suitability and usability were considered in a lesser capacity. In the design phase, the architectural changes will be made for a limited number of 'use cases' that can then be used as showcases of how to model the new architecture for the rest of the system.

The scientific relevance of this work regards the architectural styles for LAN-based web applications and services that can be used in mission- and safety-critical systems, a compilation of product quality risks in a military and/or mission- and safety-critical context, and how migration plans can be used to make decisions about partial implementations of a system. The relevance for practice is for the industry to have guidelines into how their own mission- and safety/critical systems can be adjusted to include LAN-based web applications and services. The relevance for Thales specifically is to have a formal architectural description of its current TACTICOS system and the context surrounding it; a blueprint for a new architecture for TACTICOS that incorporates web applications & services.

[2] ISO/IEC 25010 standard - product quality model: functional suitability; reliability; performance efficiency; usability; maintainability; security; compatibility; portability (as cited by (Wagner, 2013)).

1.4 Research Methodology

The theoretical framework for this research is based on architectural styles and product quality risk and how these subjects are influenced by a military and/or mission- and safety-critical context. For the design part of the research, the focus is on architectures and migration planning.

The research questions are based on the steps of the Design Science Research Methodology. This methodology was designed to be consistent with earlier knowledge about design science in information systems literature, and provides a process for design science research (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). The relations between the research questions of this thesis and the steps of the DSRM process are shown in Figure 1. The first step, 'problem identification and motivation', is covered by RQ1. Research questions 2 through 4 cover the second step of the methodology: 'define the objectives for the solution'. The third step 'design and development' is done through RQ 5 and 6. The steps of 'demonstration' and 'evaluation' are covered by research question 7.

To answer research questions 1 and 2 a literature search was done to find theories/frameworks to structure respectively the cataloguing of problems in the current architecture and the gathering of requirements/constraints for the architecture to be designed. In both cases, this was followed by a search through internal documentation from the company that initiated this project and interviews with company experts on the subjects. For both questions 3 and 4 a covering overview of their respective subjects was needed to be done, thus this was done by a systematic literature review. To answer question 3, the pros and cons of the architectural styles were also identified and analysed to be able to choose architectural styles to be used in the design phase.

To answer question 5, we began with applying the chosen architectural styles to the case-studies' context it will operate in, after which the architecture itself was developed, and this was done for some use cases. We answered question 6 by performing a literature search into best practices of architectural migrations, after which the migration path for the architectural changes was designed. To answer question 7 an evaluation of the design and several prototypes was done.

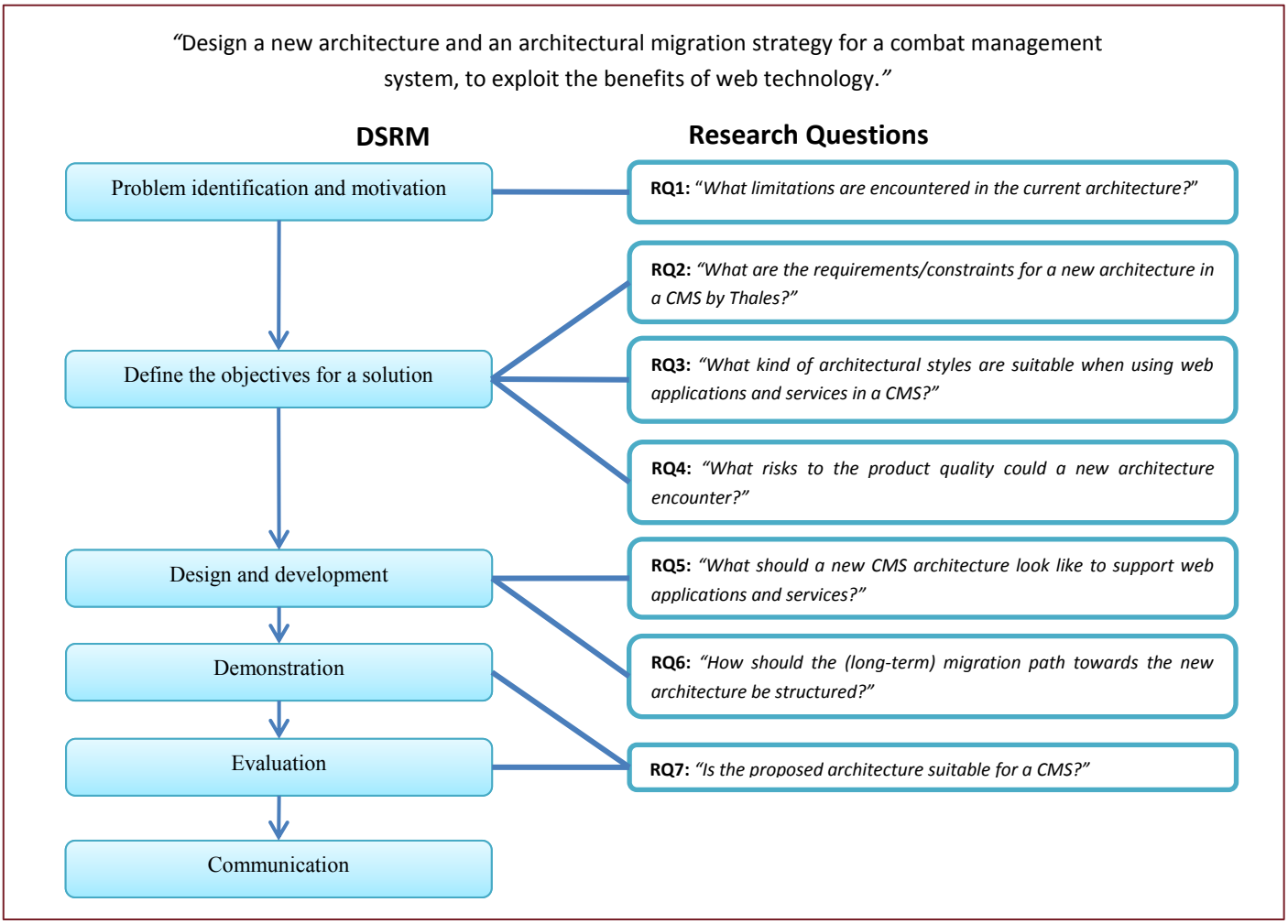


Figure 1: The relations between the research questions and the steps in the DSRM process

1.5 Thesis Structure

The thesis is further structured as follows:

Chapter 2 - contains an analysis of the problem and its context: it contains the answers to research questions 1 & 2. First the downsides of the current architecture are given. In addition, an analysis of the requirements and constraints Thales places on a new architecture is shown here, to represent a viewpoint typical for these kinds of combat management systems, as these requirements and constraints influence architectural choices and designs.

Chapter 3 - discusses the current architecture of the combat management system in the Thales case study that was modelled in Archimate diagrams as an example of a system that can be upgraded with web technology.

Chapter 4 - reports on the results of a literature search regarding research question 3 that looked at the different architectural styles that can be employed in a new architecture for mission- and safety-critical systems.

Chapter 5 - Discusses the design of the target architecture, as well as the migration strategy between the 'as-is' and the 'to-be' architectures, where different parts of the architectural migration are combined in 'transitions' which can be used in the decision process leading up to an actual implementation of the designed architecture. This chapter answers research questions 5 & 6.

Chapter 6 - gives an evaluation of the designed architecture and migration path to assess their validity against the measures of maintainability, flexibility, upgrade- and expandability, and mobile capabilities.

Chapter 7 - gives the conclusions that can be drawn from this thesis, how the research questions were answered, and the recommendations given for the company and future work in the scientific field surrounding this research.

Thesis Chapters	Research Questions	DSRM
Chapter 1 - Introduction		
Chapter 2 - Problem Analysis	**RQ1**: *What limitations are encountered in the current architecture?*	Problem Identification and motivation
	RQ2: *What are the requirements/constraints for a new architecture in a CMS by Thales?*	Define the objectives for a solution
Chapter 3 - Modelling The "As-Is" Architecture		
Chapter 4 - Architectural Styles	**RQ3**: *What kind of architectural styles are suitable when using web applications and services in a CMS?*	
Chapter 5 - Design	**RQ4**: *What risks to the product quality could a new architecture encounter?*	
	RQ5: *What should a new CMS architecture look like to support web applications and services?*	Design and development
	RQ6: *How should the (long-term) migration path towards the new architecture be structured?*	
Chapter 6 - Evaluation	**RQ7**: *Is the proposed architecture suitable for a CMS?*	Demonstration
		Evaluation
Chapter 7 - Conclusions & Recommendations		

Table 2: Relations between thesis chapters, research questions, and the DSRM

Chapter 2 - Problem Analysis

2.1 Case study context

To look at an implementation of how web technology can be used in a military and safety critical system a case study is done with the Combat Management System TACTICOS by Thales Netherlands. This case study is deemed a viable representation of the characteristics of mission- and safety-critical system, and specifically of combat management systems. This case study is helpful in identifying challenges and opportunities when introducing web technology in such systems.

Thales is a multinational focused on providing security solutions, in the following markets: Aerospace, Space, Ground transportation, Defence, and Security. Their revenues over 2014 were € 13 billion, and they have 61,000 employees spread over 56 countries (Thales Group, 2015). TACTICOS is a Combat Management System (CMS) made by the Naval Application Engineering department of the Dutch Thales subsidiary: 'Thales Netherlands'. A combat management system integrates the weapons and sensors of a naval vessel (Deelstra, Sinnema, & Bosch, 2005). In the operations room several operator consoles (along with electronic conference tables and collaboration screens) offer a unified human machine interface that adapts to the operational tasks of each operator, anticipating the operator's workflow to give relevant information and control options at the right time from across the different systems of the vessel (Thales Nederland, 2013). TACTICOS has three functional segments: Command support, providing decision support for the medium to long term planning of missions; Command & Control, providing real-time situational awareness and controlling operational assets; and Combat execution, using the ships (weapon) systems to deal with threats (Gething, Williamson, Fuller, & Ewing, 2014) (Thales Nederland, 2008).

2.2 Approach

In this chapter, two of the research questions are answered, the first research question ("*What limitations are encountered in the current architecture?*") is used as the first step in the Design Science Research Methodology, "Problem identification and Motivation" (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). In section 2.3 we analyse the problems that drive the need for a new architecture, but we also explore other problems that can be solved when creating a new architecture.

The second research question ("*What are the requirements/constraints for a new architecture in a CMS by Thales?*") is one of the parts in this research to cover the second step in the DSRM process, "Define the objectives for a solution". In subsection 2.3, next to the investigation into the downsides of the current system a study into the context of the problem was also done, by researching what requirements and constraints are placed on a new architecture by the development group of TACTICOS and other parts of Thales, to reveal the boundaries in which the new architecture needs to be placed.

To answer both research questions a twofold approach is used: first a literature search was done through internal company documentation, to find written accounts of current problems with TACTICOS, information about the current architecture, and to find formal requirements and constraints placed on the new architecture. But as the documentation regarding this subject was insufficient, inaccurate, and not up-to-date, this was complemented by interviews with Thales employees directly working on the development of TACTICOS, employees surrounding the TACTICOS project, and employees from other areas of Thales that could provide insight into what needs to be taken into account when designing a new architecture for combat management systems, and TACTICOS in particular.

2.3 Interviews with experts

To give structure to the interviews, several concepts relating to the problems of IT systems/architectures and requirements/constraints that can be placed on an architecture, are used. The following four groups of constructs were used to structure the downsides of the current architecture: Several properties of software systems in which problems can exist are given by Brooks (Brooks, 1987): complexity, conformity, changeability and invisibility. Byrd & Turner (Byrd & Turner, 2000) give several constructs relating to the flexibility of an IT infrastructure: connectivity, compatibility, application functionality and data transparency. The COBIT framework gives several criteria that business information needs to adhere to (as cited by (Guldentops, 2002)): effectiveness, efficiency, confidentiality, integrity, availability, and compliance. The ISO/IEC 25010 quality model for quality in use (as cited by (Wagner, 2013)) gives the following constructs to determine the quality of interactions with a system in use: satisfaction, effectiveness, freedom from risk, efficiency, and context coverage. To structure the search for requirements and constraints for the new architecture two groups of constructs were used: There are several sources from which design constraints can come that can restrict the boundaries of a solution (van den Berg, Tang, & Farenhorst, 2009): functional requirements, quality requirements, Business & IT strategy, System goals, Industry standards, Project resources, Contractual obligations, bargaining power, resistance to change, political pressure, technical system environmental, business practices, and legislation. The ISO/IEC 25010 standard provides a product quality model that can be used to define the requirements for a system (as cited by (Wagner, 2013)): functional suitability, reliability, performance efficiency, usability, maintainability, security, compatibility, and portability.

This gives a purposely broad range of constructs that are used to trigger the interviewees to try to give an as covering as possible answer to the research questions in this chapter, even though the scale of the interviews is kept small due to time and resource limitations. The range of these constructs is not fully covered in each interview, but due to the heterogeneous nature of the group of interviewees, certain constructs most in line with their expertise were highlighted. Not all constructs are equally useful to what is needed to be known in relation to the nature of the TACTICOS system. That is why weights have been given to each of the constructs, to take priority in the interviews. The interviews are semi-structured, starting with some general directed questions towards some of the important constructs, and drilling deeper with more open questions based on the answers given.

To gather the requirements the interviews were done with 5 employees of Thales, they came from 4 of the asset teams that work on TACTICOS and 1 was from another department (see Figure 2 below). The interviewees were notified of the purpose of this research and the goals of these interviews, which were to find out what was wrong with the current architecture, and what the boundaries for a new architecture would be. With these goals in mind, the interviews were done following the previously mentioned constructs as guidelines. After each interview was done, the notes from the interview were sent to the interviewee to confirm whether the notes reflected what they meant in the interview. During the first few interviews, the interviewees not only came up with problems of the current architecture and boundaries for a new architecture. They also came up with positive points of the current architecture that they would like to see perpetuated into a new architecture. They came up with opportunities that web technologies could create for the TACTICOS system. Moreover, they came up with what you should watch out for with web technologies. These were useful additions to the interview process, and thus were incorporated into the remaining interviews. It was deemed unnecessary to redo the first interviews with the additional goals in mind, as these were the interviewees that already came up with the additional suggestions themselves.

		Departments								
		Tacticos								
		Product asset A	Product asset B	Product asset C	Product asset D	Product asset E	Product asset F	Product asset G	Product asset H	Other departments
Name	**Function**									
Employee 1	Software Engineer		x				x			
Employee 2	Software Architect							x		
Employee 3	Software Architect								x	
Employee 4	Software Engineer									x
Employee 5	Architect							x		

Figure 2: (Anonymised) table of the interviewed employees and their coverage of the departments

To rework the notes from the interviews and some internal documents (that related to downsides of the current architecture and requirements/constraints for a new architecture) into useful requirements, several steps from the grounded theory for requirements gathering as described by (Halaweh, 2012) were used. Firstly by marking important ideas and suggestions in the notes of the interviews and in the internal documents used. With the next step of Open Coding the ideas and suggestions were reworked into codes (in vivo coding, as the codes were kept close to the terms used by the interviewees and in the documents). When that was done, the concepts were related to each other and across interviews (and documents) in the axial coding step, to see general concepts emerge.

To structure the concepts that emerged, an adapted form of SWOT-analysis was used. This choice was based on the different kind of suggestions that were given by the interviewees and the assumption of general familiarity in businesses with SWOT-analyses. Were a traditional SWOT-analysis gives the internal strengths & weaknesses of a business or project, and the threats and opportunities of the external environment (Hill & Westbrook, 1997), here the SWOT-analysis was adapted to describe the strengths and weaknesses of the current architecture, and the opportunities and the threats for a new architecture. Were the ‘strengths’ are positive points of the current architecture that should ideally be taken along into the new architecture. The ‘weaknesses’ are the negative points of the current architecture that should ideally be circumvented in a new architecture. The opportunities are the ways in which new technologies (or a change in architecture) could benefit the system. The threats are the potential problems that could present themselves in a new architecture (specifically due to web technologies).

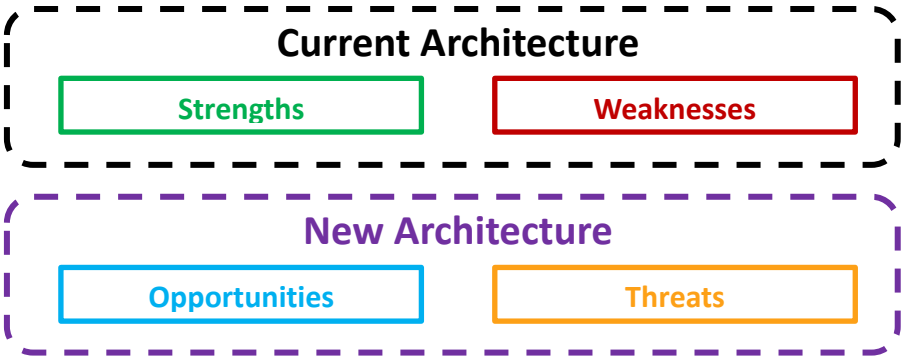


Figure 3: Adapted SWOT-analysis based on 'current architecture vs. new architecture' instead of 'internal vs. external'

The (aggregated) results from the interviews are shown hereunder, divided into the 4 categories of the adapted SWOT-analysis:

2.3.1 Strengths of the current architecture

These are some of the positive points attributed to the current system, which have proven to be useful, and should ideally be taken along into a new architecture:

STR-01: OpenSplice[3] decouples applications from each other. Because of the Publish/Subscribe pattern used in the OpenSplice software the communication between applications isn't direct, but is decoupled in time. This way applications can take up a task when they have the time and resources to do that task

STR-02: Testing is done a lot, guarding the quality of the product. Within Thales a lot of product testing is done, on different testing levels, which guards the quality of the product.

2.3.2 Weaknesses of the current architecture

These are some of the negative points attributed to the current system, which hold the system back, and should ideally be circumvented in a new architecture:

WKN-01: Java Swing is old, complex and causes other problems. The technology used for the HMI of the TACTICOS system, Java Swing, is outdated. Causing requested functionality to lead to too much complexity and costs.

WKN-02: There is a large dependence between the HMI and the code behind. Due to the outdated Java Swing HMI technology changes in the HMI necessitate changing the code behind the HMI, causing increased overhead in development costs.

WKN-03: The Fat-client architecture is bloated. On the client side, where the system is used, too much processing has to be done. Besides this, a lot of software has to be installed on each operator console.

WKN-04: There is too much integration/dependency between applications. When one application fails there is a large chance that it will take other programs down with it, because there are too many dependencies between them.

[3] Vortex OpenSplice Data Distribution Service: www.prismtech.com/vortex/vortex-opensplice

WKN-05: Usability / User Experience has a low priority. HMI screens are now usually designed based on what data there is available from the different systems, but not enough thought is given to what the operator really needs from his viewpoint.

WKN-06: There is no unified look for the HMI. Because the system is now a combination of gradually reworked applications/systems, there are a lot of mixed looks and mixed techniques causing questions from customers.

WKN-07: OpenSplice uses a global datamodel. The OpenSplice Data Distribution Service uses a global data model, which means that when new data needs to be introduced, the whole datamodel on all connected systems has to be reworked to incorporate the new data definitions.

WKN-08: There is not enough focus on security within the network. Even though the network is protected from outside attack, not enough security measures are taken within the system against attacks/malware from within.

WKN-09: The data in OpenSplice is not protected from unwarranted access. Everyone on the system can access the data transmitted through the data distribution system.

2.3.3 Opportunities for a new architecture

The following points are some opportunities that were thought to be relevant when changing architectures, and especially when changing to an architecture including web technologies:

OPP-01: An architecture supporting a testing strategy. Designing a new architecture creates a good opportunity to incorporate architectural structures that provide facilities for thorough testing. Also there are good 3rd party testing frameworks available for web development.

OPP-02: A modular approach. Web technology is by its nature relatively modular, if one incorporates this into a new architecture applications can get more decoupled to prevent cascading failures.

OPP-03: Deploying web technologies on a system by system basis. Due to the modular nature of web technologies, the development of new systems to replace older systems can be done incrementally and can coexist along older systems.

OPP-04: Mainstream development technologies. When one keeps the used development techniques more mainstream, one can use more Commercial-of-the-Shelf 3rd party software, instead of having to develop it oneself. Besides this there are more/cheaper personnel to be found, when not using obscure development technologies.

OPP-05: Expansion to mobile devices. Web technologies are hardware independent, and are now being designed with mobile devices in mind. Thus this technology can be used to extend the system to work on mobile devices aboard the ship.

OPP-06: Web technologies can make you less dependent on a continuous connection to the server. As web technologies are designed for the internet where connections can be dropped, they are built to be more resistant to these connection drops than conventional direct server-client connection techniques.

OPP-07: Separation of concerns in the HMI. When there is more separation between interface design, its behaviour, and the visual design, these areas can be developed and updated independently from each other.

OPP-08: Better opportunities for prototyping. If the HMI layer can be kept thin, this layer can be changed more easily, and thus prototyping can be done more rapid and more often, resulting in better HMI's.

OPP-09: Resolution independent HMI. Web layout techniques can be made responsive and resolution independent, so current problems with per-pixel design on higher DPI screens can be prevented.

2.3.4 Threats to a new architecture

The following points were thought to be a threat to the system when changing architectures, and specifically to an architecture involving web technologies:

THR-01: System life time & 3rd party library support. Systems like TACTICOS are deployed on ships for life times of 15 to 20 years, and the system needs to be maintained for that life time. This means that during that time development itself and the development environment needs to be available for that time period. This considering that the ecosystem surrounding web technologies refreshes itself very fast, technologies come up fast, and disappear quickly.

THR-02: 3rd party components can cause integration issues. Incorporating 3rd party (web) libraries can cause unwanted interactions within the system.

THR-03: The maintainability of contemporary web technologies. Techniques like JavaScript are not compiled, so mistakes in development may not be noticed until testing or even until runtime.

2.4 Requirements for the new architecture

The results from these interviews were then analysed to combine and genericize them into the following set of requirements (see Table 3 below) that were to be kept in mind for the design of a new architecture later on in this thesis.

REQ-01 is defined as 'The system should be modular', it came from a combination of the, in section 2.3 proffered, opportunities OPP-02, OPP-03, and OPP-07. **REQ-02**, 'Changes in functionality should not require the entire system to be updated.', follows from Weaknesses WKN-01, WKN-02, and WKN-07; the opportunities OPP-03, OPP-07, and OPP-08. **REQ-03**, 'Applications should be decoupled from each other, so tasks can be done when there are resources available, and failures don't cascade to other applications.', Is constructed based on the strength STR-01; the weakness WKN-04; the opportunities OPP-02, OPP-06; and the threat of THR-02. **REQ-04**, 'There should be less dependency on a continuous connection between client and server.' came from strength STR-01; and opportunity OPP-06. **REQ-05**, 'The technology used for the GUI should be modern and easy to make changes to.', follows from weaknesses WKN-01, WKN-02, WKN-05, and WKN-06; and opportunities OPP-07, OPP-08, and OPP-09.

REQ-06, 'Prototyping should be rapid and done often.', consists of opportunities OPP-07 and OPP-08. **REQ-07**, 'Processing should be done more on the server-side instead of on the client-side.', is constructed with the following in mind: Weakness WKN-03; and opportunity OPP-05. **REQ-08**, 'There should be more of a focus on security within the network.', deals with weaknesses WKN-08 and WKN-09. **REQ-09**, 'Data should be protected from unwarranted access within the network.', is related to WKN-09. **REQ-10**, 'Expansion to mobile devices should be supported.', is constructed based on opportunity OPP-05.

REQ-11, 'The system needs to be maintained for 15-20 years.', is based on threat THR-01. **REQ-12**, 'Applications/systems should be designed for testability.', is a combination of strength STR-02; opportunities OPP-01 and OPP-07; and threats THR-02 and THR-03. **REQ-13**, 'Use technology that is mainstream/common, so 3rd party components and personnel are easier to be found.', deals with weakness WKN-01; opportunity OPP-04; and threat THR-01. **REQ-14**, 'The deployment of a new architecture should be done incrementally and co-exist with legacy technology.', deals with weaknesses WKN-02, WKN-04; and opportunity OPP-03.

Requirement	Description
REQ-01	The system should be modular.
REQ-02	Changes in functionality should not require the entire system to be updated.
REQ-03	Applications should be decoupled from each other, so tasks can be done when there are resources available, and failures don't cascade to other applications.
REQ-04	There should be less dependency on a continuous connection between client and server.
REQ-05	The technology used for the GUI should be modern and easy to make changes to.
REQ-06	Prototyping should be rapid and done often.
REQ-07	Processing should be done more on the server-side instead of on the client-side.
REQ-08	There should be more of a focus on security within the network.
REQ-09	Data should be protected from unwarranted access within the network.
REQ-10	Expansion to mobile devices should be supported.
REQ-11	The system needs to be maintained for 15-20 years.
REQ-12	Applications/systems should be designed for testability.
REQ-13	Use technology that is mainstream/common, so 3rd party components and personnel are easier to be found.
REQ-14	The deployment of a new architecture should be done incrementally and co-exist with legacy technology.

Table 3: Requirements for the new architecture

Chapter 3 - Modelling The "As-Is" Architecture

Architectures are high-level descriptions of IT systems represented as interconnected smaller applications. An architectural description is made to give structure to the understanding of the IT applications, as a framework to solve current and future problems (Britton & Bye, 2004). An integral purpose of architectures is to show the motivation behind the choices that are made, to justify why the different systems need to work together. Architectures contain three kinds of elements: processing elements, wherein the actual work takes place; data elements, which describe the information on which the processing is done; and connecting elements, which define the way the different processing and data elements are connected (Perry & Wolf, 1992). An Architecture Description Language (ADL) is a modelling language used to describe an architecture: Archimate is an example of an ADL for broad use (ISO/IEC/IEEE, 2011). It provides a notation to describe the relations of an architecture within and across the business, application, and technology domains. The original Archimate was created in 2002-2004 in the Netherlands and ownership was later transferred to The Open Group. It became an Open Group standard in 2009, and version 2 of the standard was released in 2012. Next to the core ADL, it got a 'motivation extension' to model the motivations of stakeholders and their requirements, and the 'implementation and migration extension' to model the way to implement a new architecture (Josey & Franken, 2012). In this thesis, the Archimate 2.0 ADL was used to model the architectural descriptions as described in the book by Iacob et al (Iacob, Jonkers, Quartel, Franken, & van den Berg, 2012).

Several ways were used to gather the information necessary to create an architectural description of the current system: internal documents which contained architectural descriptions of parts of the system or other kinds of descriptions of the system; informal talks with several employees involved with the development of the TACTICOS system to gain an understanding of the workings of the system and the components within; Incremental reviews and updates of the model by showing it to several people and gathering feedback from them, which was used to incrementally refine the model. Due to the large amount of different subsystems that can be integrated with TACTICOS, and the fact that each solution can be a combination of different subsystem integrations, only a few subsystems were chosen to be included in the model of the current architecture for this research. As these were deemed to be representative for most of the different possibilities in which the components of the system interact with each other.

Over the next pages several models representing (parts of) the TACTICOS environment will be shown, to create an understanding of the system, and be able to envisage a new architecture that includes web applications.

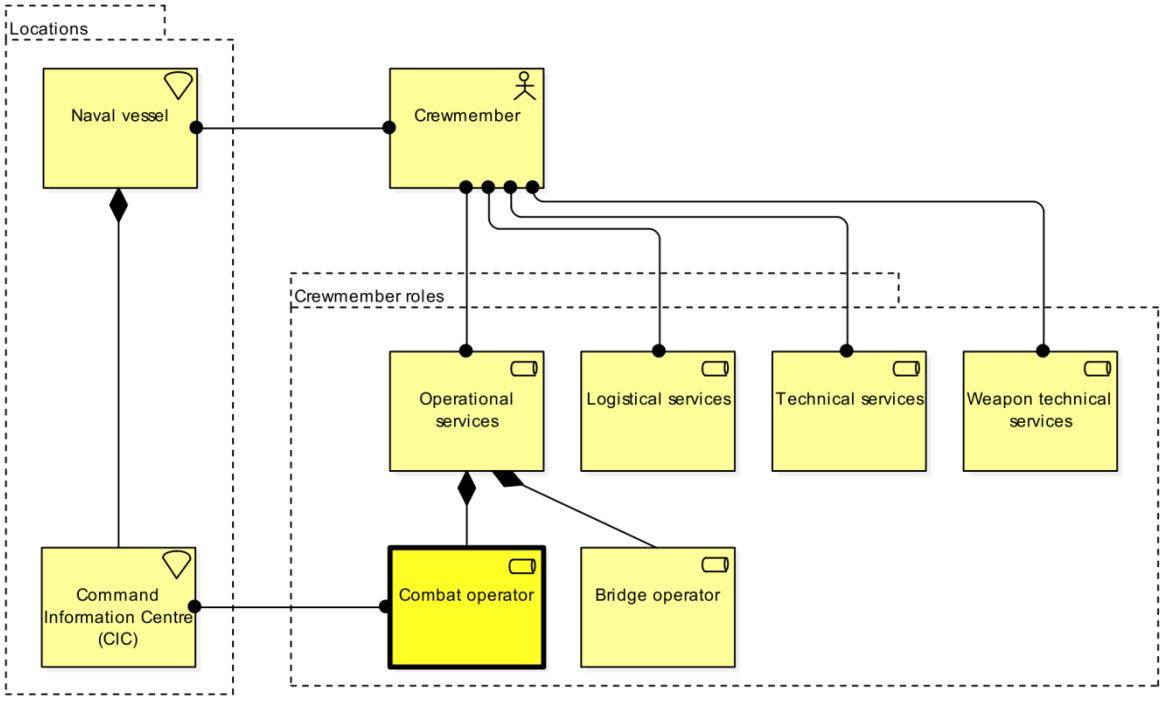


Figure 4: A selection of crewmember roles within a naval vessel

On a naval vessel several crewmembers each have their own roles aboard the ship. These roles have the main categories of operational services, logistical services, technical services, and weapon technical services. Within that first category of operational services, two subcategories can be found: Combat operators and bridge operators. With the latter roles in charge of steering the ship and other navigational tasks. The former roles, the combat operators, are in charge of the sensing, combat execution and operational planning tasks of the vessel from within the Command Information Centre (CIC). These operators are using the TACTICOS system to perform their tasks.

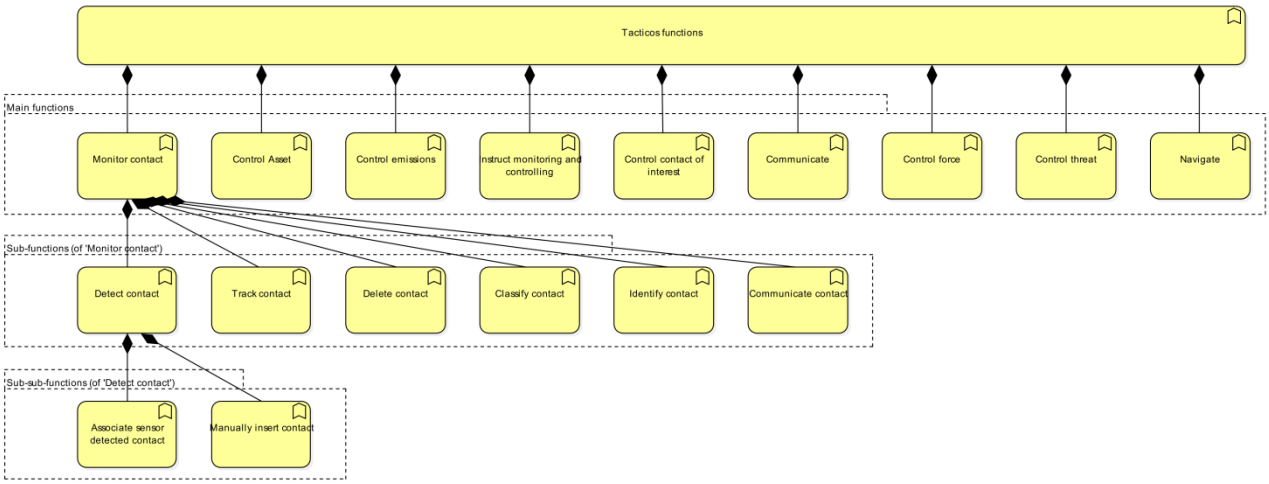


Figure 5: A selection of functions that can be performed with the Tacticos system

The TACTICOS system is a fairly complex and expansive system. To illustrate this Figure 5 gives a small subset of functions that can be performed with the system by the operators. All 9 main functions are shown in this

diagram, but each of these main functions has sub-functions and sub-sub-functions. As an example the sub-functions of the first main-function, 'Monitor contact', are shown. And below that the two sub-sub-functions of the 'Detect contact' sub-function are shown. Each of the sub- and main-functions has one or multiple functions under it. This comes to a total of 128 operational functions. This to indicate the complexity of the system, and the need to be selective in which parts of the system to model in this thesis.

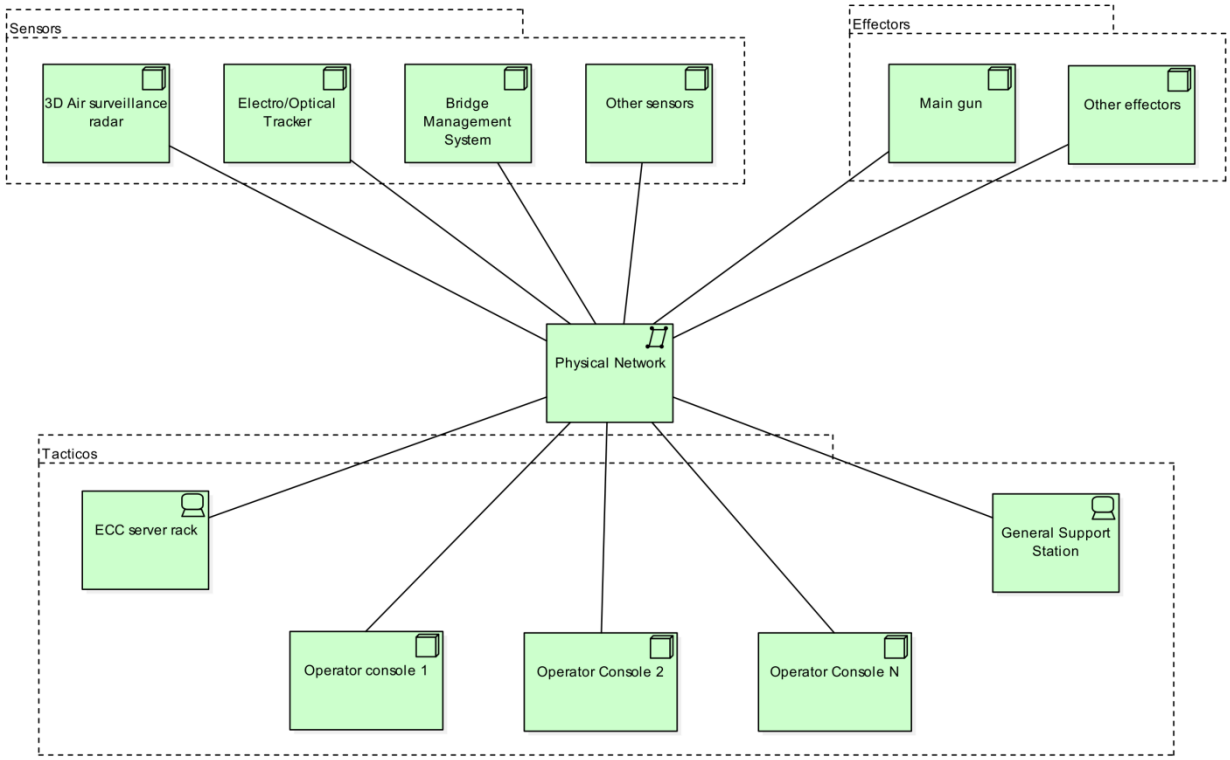


Figure 6: Global overview of a selection of the (sub)systems.

In a vessel with a TACTICOS solution several systems are coupled together. As said previously only a selection of systems are shown in these architectural models, as including more would make the models unnecessarily complicated. In Figure 6 the different systems (shown as infrastructural computational nodes and devices) are categorized in three different categories: Sensors: The subsystems that gather information about the environment, like: search radars, tracking radars, navigational radars, communication links, sonar, etc.; Effectors: The subsystems that can sort an effect on the environment, for example: the main gun, missile bays, Electronic countermeasures, etc.; TACTICOS: The systems in this category are what make up the TACTICOS system itself, they interconnect the different sensors and effectors and provide the link between the ship's (combat) systems and the operators; All these systems are associated with the same physical network, that spans the vessel.

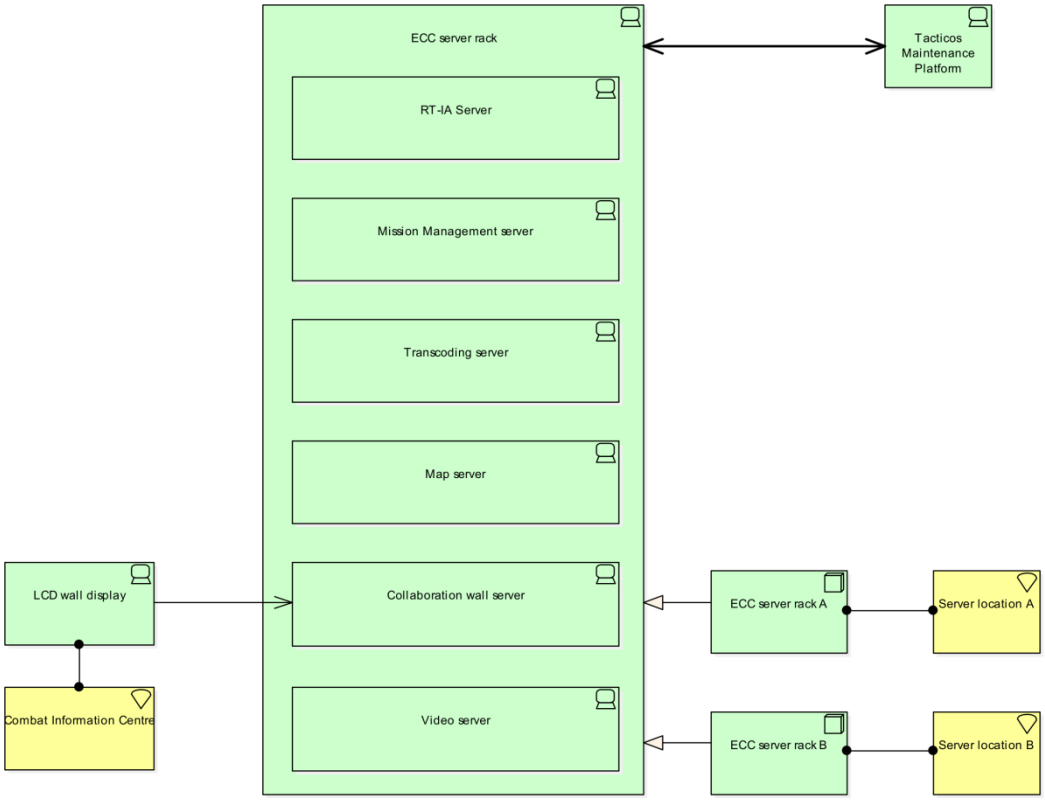


Figure 7: Contents of an Environmental Conditioning Cabinet server rack

In Figure 7 the contents of the Environmental Conditioning Cabinet (ECC) server rack are shown. In this model 6 different servers contained in the server rack are shown. Per solution and deployment different servers are selected to be included in the ECC, based on the operational needs. For redundancy there are usually 2 ECC's deployed on a ship, placed in different locations of that ship. The different servers are then either spread over the 2 ECC's or duplicated (duplicated both for redundancy and for upscaling of processing power). In this model the following kinds of servers are included: Real Time – Interface Applications server: an application server that provides processing for tasks related to the interaction with sensor and effector subsystems; Mission Management server: delivers processing and storage for the operational planning of missions (supporting the Command Support and Command & Control functional segments of the Combat Management System, as opposed to its Combat Execution segment); Transcoding server: provides video transcoding capabilities to change video formats. Map server: serves different kinds of maps and sea charts. Collaboration wall server: this server displays images, video, console screens and other information on large displays in the Combat Information Centre, where the operator consoles are located, for shared situational awareness. Video server: for the storage and streaming of video sources from the sensor subsystems.

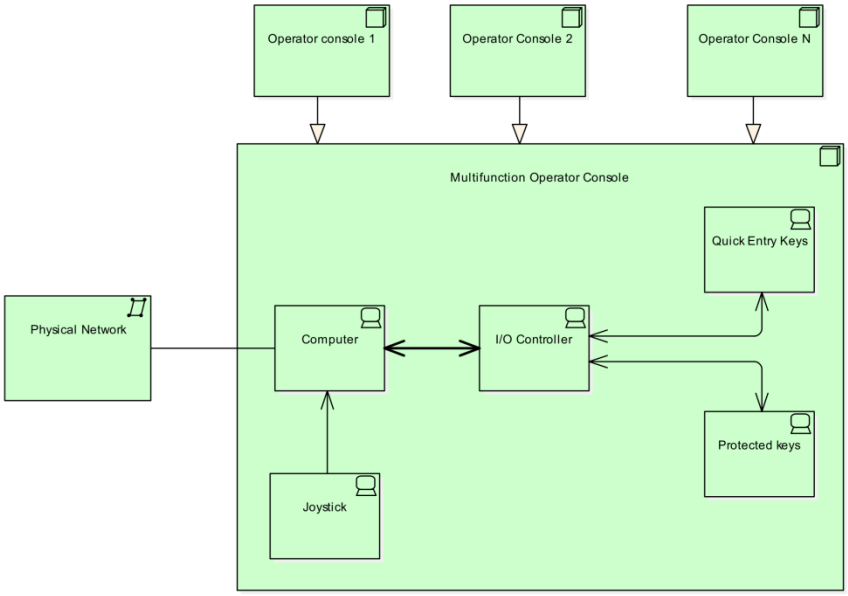


Figure 8: Overview of a Multifunction Operator Console

The Combat Information Centre (CIC) of the vessel contains between one and several so-called Multifunction Operator Consoles (MOC). On these consoles the combat operators work with the TACTICOS system to control the ship's (weapon) systems, plan missions, and train with simulated scenarios. Each console is essentially the same, but the TACTICOS software adapts itself to the role the logged-in combat operator has, giving for example an Anti-Air Warfare (AAW) work set for the Anti-Air Warfare Officer (AAWO), an Electronic Warfare (EW) work set for the Electronic Warfare Coordinator (EWC), as well as specialized work sets for the Principal Warfare Officer (PWO) or the Commanding Officer (CO), and so on. The MOC consists (see Figure 8) of a network-connected computer with screen, trackball & Keyboard, as well as a PTZ control joystick for video control. The computer is also connected with a network cable to an I/O controller device that interacts with the Quick Entry Keys (QEK) and the Protected Keys (PK). The QEKs are configurable hardware buttons with small dedicated LCD screens to display the current function of each button, which can be programmatically changed based on the current work set and state of the TACTICOS system on the MOC. The PKs are similar to the QEKs but have protective covers over the buttons to prevent accidental activation.

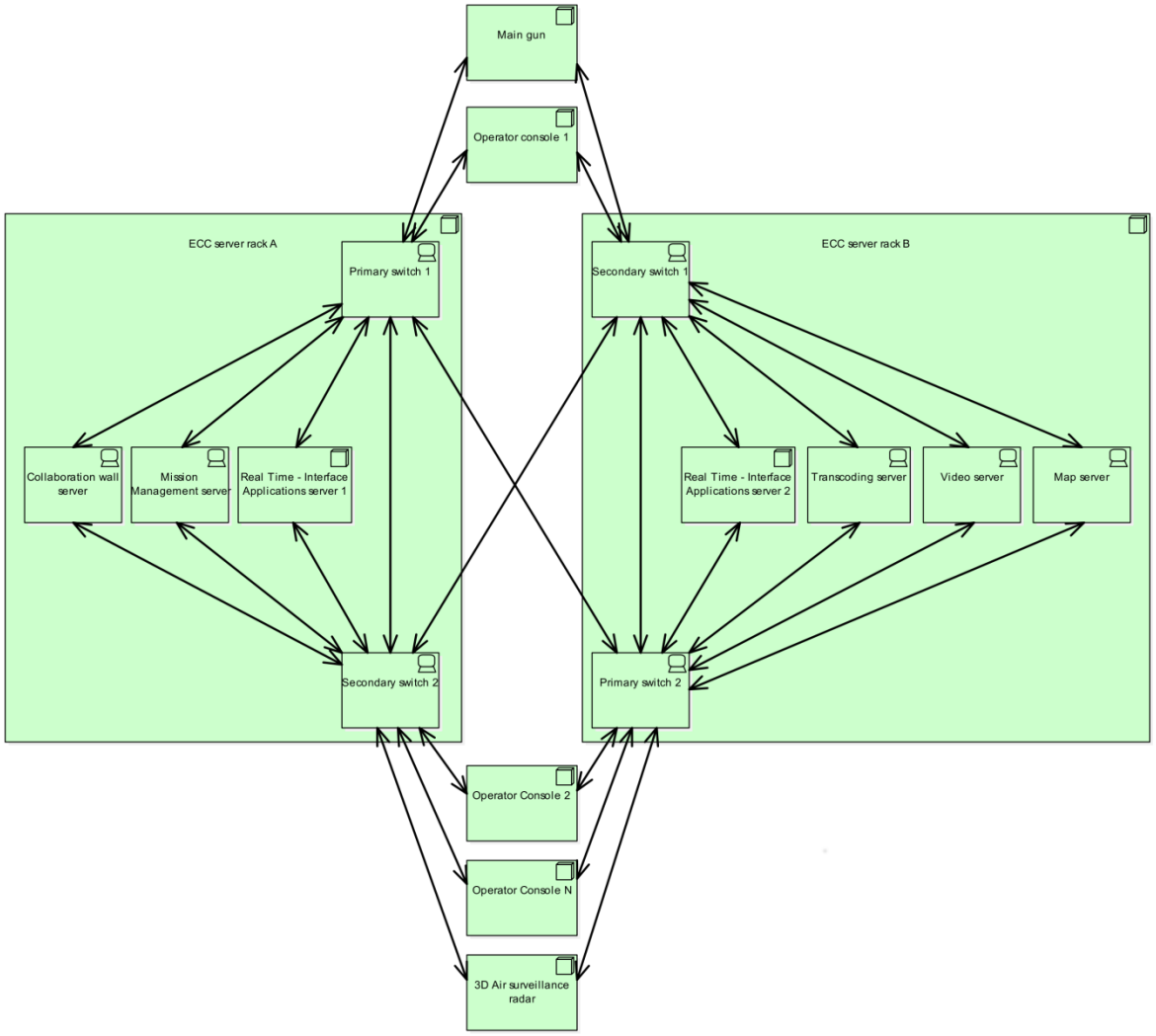


Figure 9: (Physical) network layout for redundancy.

In Figure 9 the way to create network path redundancy in the physical network is shown. In this so-called 'quad-star' configuration 4 network switches are spread over 2 ECC server racks. In the ECC A the primary switch of network 1 is connected to the secondary switch of network 2, in ECC B the secondary switch of network 1 is connected to the primary switch of network 2. Between the ECC's both primary switches are interconnected to each other, and both secondary switches also to each other. The systems outside the ECC's are then connected either to the primary and secondary switch of network 1 or the primary and secondary switch of network 2. This way all systems are still connected to each other when one of the ECC server racks is down or destroyed. The servers inside a server rack are connected only to the two internal switches.

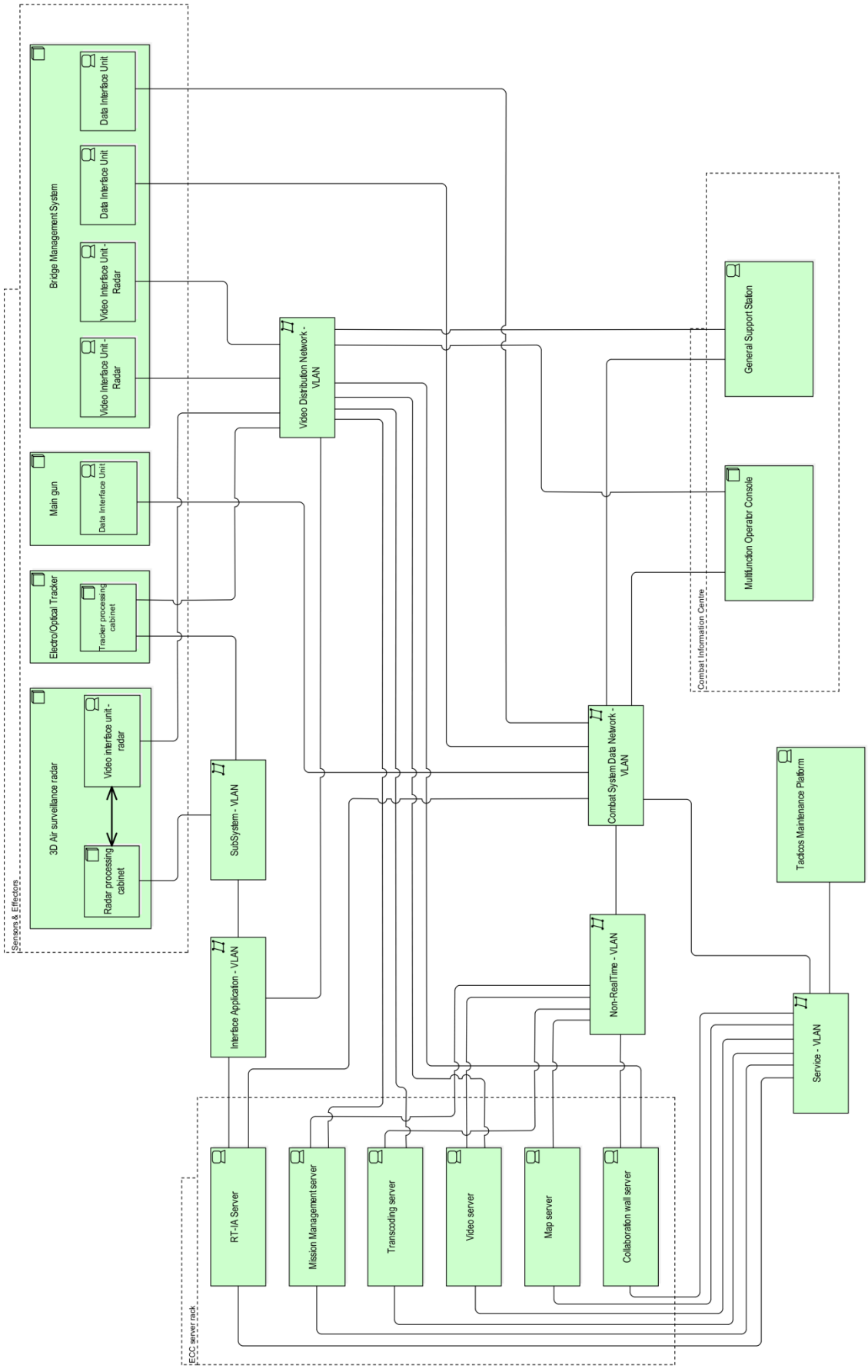


Figure 10: Interconnections between (sub)-systems and VLAN's.

In Figure 10 the different Virtual LAN's are shown that are defined for the network used in the vessel. These VLAN's are used to separate the communication between systems, and control throughput. Six different VLAN's are defined: Combat System Data Network (CDSN) VLAN, the main data network; Video Distribution Network (VDN) VLAN, the network through which video is distributed; SubSystem (SS) VLAN, for the sensors & effectors that do not have a Data Interface Unit that can communicate in the format used on the CSDN VLAN, the SS VLAN connects to the IAN VLAN; Interface Application Network (IAN) VLAN, provides the connection between the RT-IA server and the SS VLAN; Non-RealTime Network (NRTN) VLAN, for communication between servers that is not latency-critical and could interfere with the throughput of the CSDN VLAN; Service (SRV) VLAN, for servicing. Sensors & Effectors can be connected either through the SS VLAN and IAN VLAN to the RT-IA server which then provides processing and further transmission to the operator consoles, or when equipped with a Data Interface Unit that translates to the format used on the CSDN VLAN it can be connected directly to it. Radar video and TV video from sensors can be sent through the VDN VLAN by a processing cabinet or after processing by a Video Interface Unit (VIU-Radar or VIU-TV). The Multifunction Operator Consoles and the General Support Station are only connected to the CSDN and VDN VLAN's.

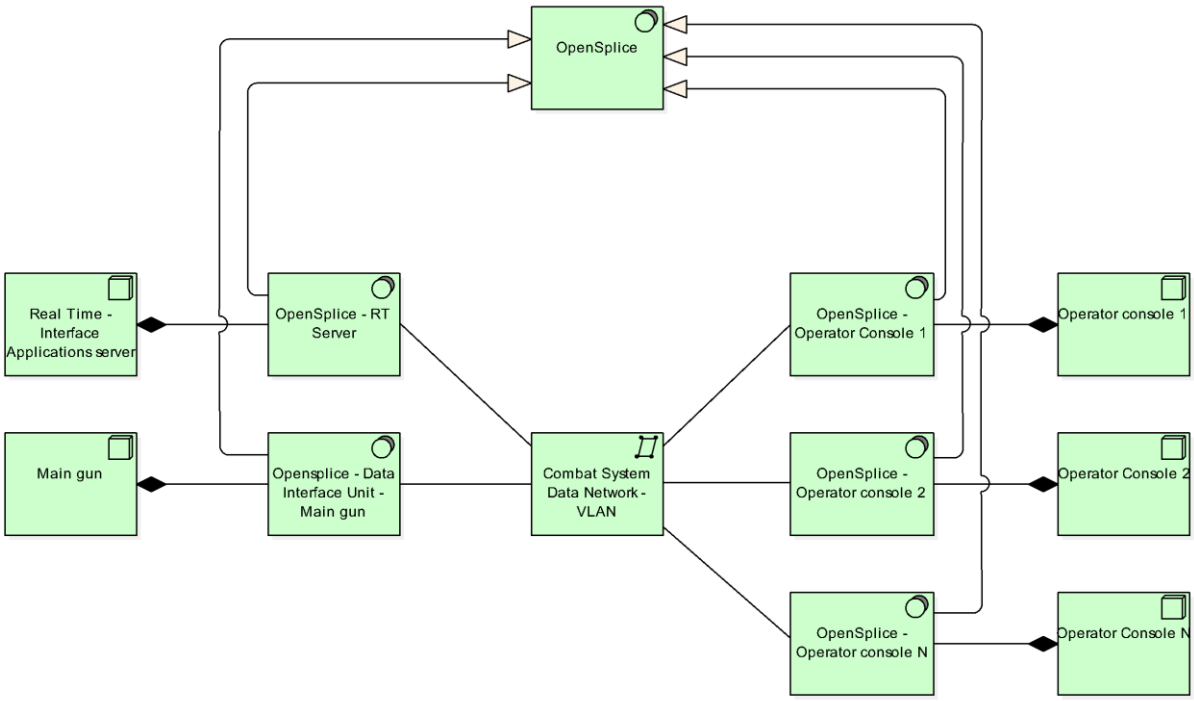


Figure 11: The OpenSplice Data Distribution Service

OpenSplice is the Data Distribution Service (DDS) that is used to transport data on the CSDN VLAN (Figure 11). The principle of a DDS is that the data is distributed between nodes based on pre-defined Quality-of-Service (QoS) rules, e.g. reliability, latency, and persistence. OpenSplice works based on a publish/subscribe model in which one of the nodes publishes information on a certain topic, which then is distributed to the other nodes running the OpenSplice application. When one of those OpenSplice nodes is subscribed to that specific topic it will then notify the application that requested that subscription and provide the data.

3.1 Use case 1: 3D search radar

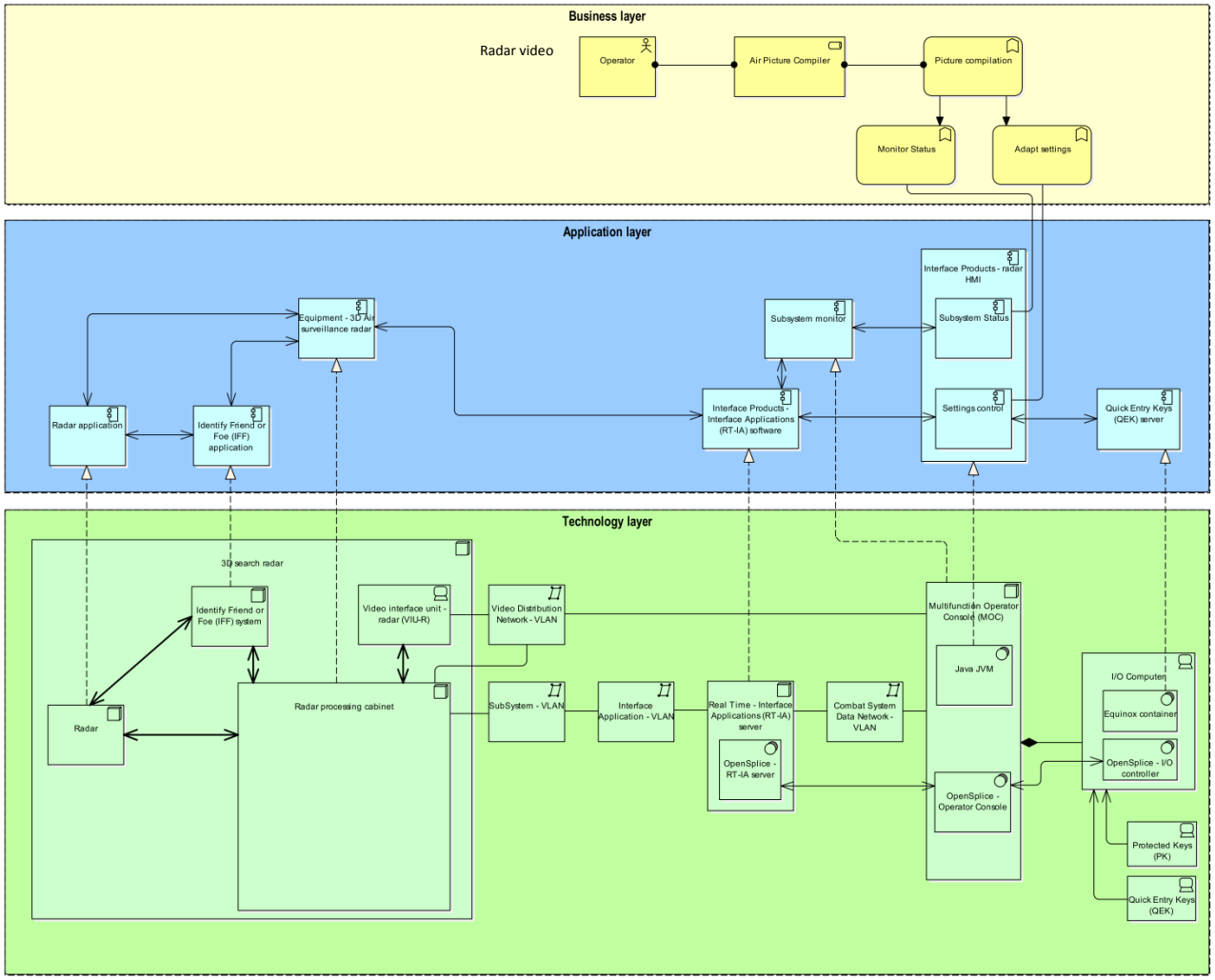


Figure 12: Application interaction between 3D Air Surveillance Radar and the Multifunction Operator Console.

The example of the 3D Air surveillance radar is used to describe how the applications between a subsystem and a MOC are connected, see Figure 12. In the radar processing cabinet an application gathers data from the radar itself and the Identify-Friend-or-Foe (IFF) system, this data is then processed and sent to the MOC. The radar video is send directly (through the VDN VLAN) to the Interface Products HMI application on the MOC. Data from the radar and IFF is send through the SS & IAN VLAN's to the RT-IA server on which further processing can be done in an Interface Products – Interface Application, after which it is sent through the CSDN VLAN to the Interface Products – HMI on the MOC. The Resource control application on the MOC is used to read the status of the radar, and to control operational state-changes of the radar system, which traverses the reverse path through the CSDN -> RT-IA server -> IAN -> SS -> Radar processing cabinet.

On the next pages the application interactions for video feeds are shown, as further examples of how applications interact in the TACTICOS system.

3.2 Use case 2: Video server

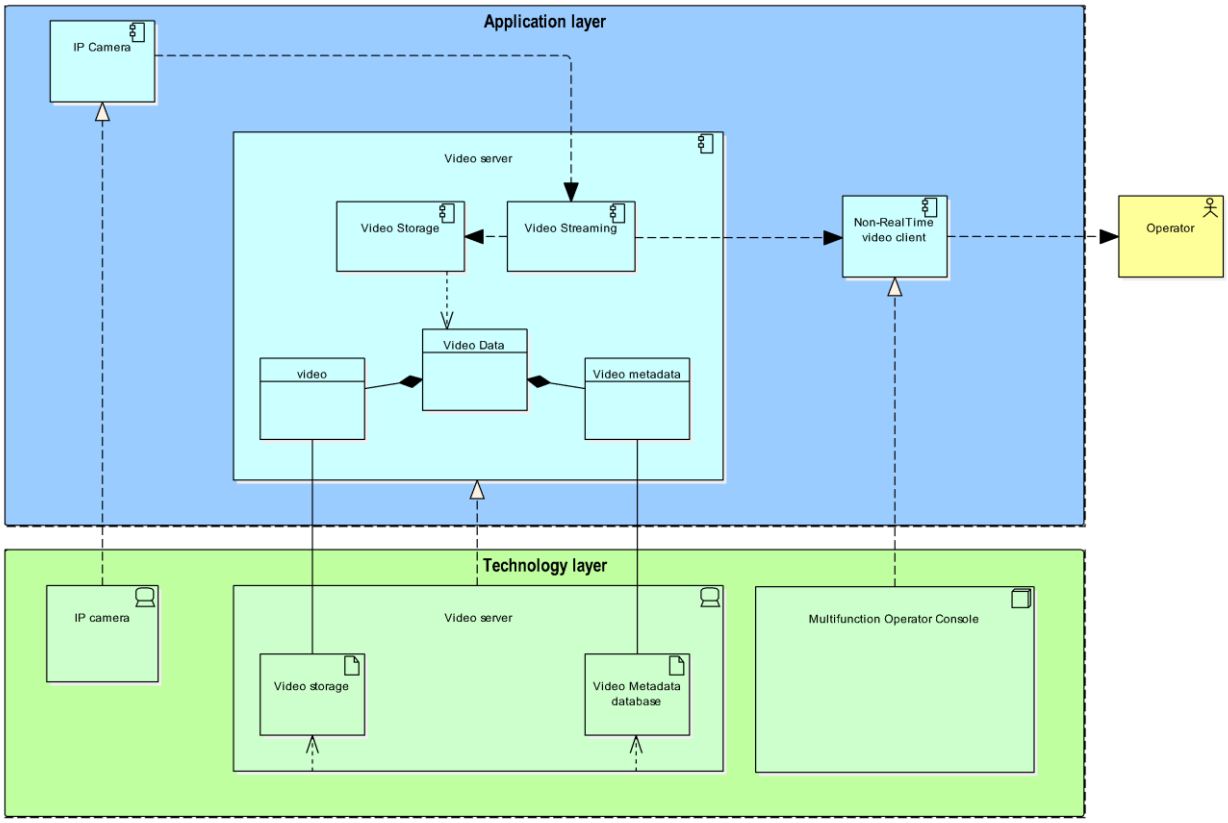


Figure 13: Application flow for non-real-time (standard latency) video.

In Figure 13 the flow of video through the applications can be seen, for cases like with IP cameras, in which no special latency requirements are needed (for an example in which there are special latency requirements see the next paragraph and Figure 14). The IP camera supplies a video feed to the video streaming application on the video server, which then streams the video through the VDN network using multicast to the non-real-time video client application on the MOC. Next to a direct feed to a MOC the video feed can also be stored, in that case the video streaming application on the video server also send the video internally to the video storage application.

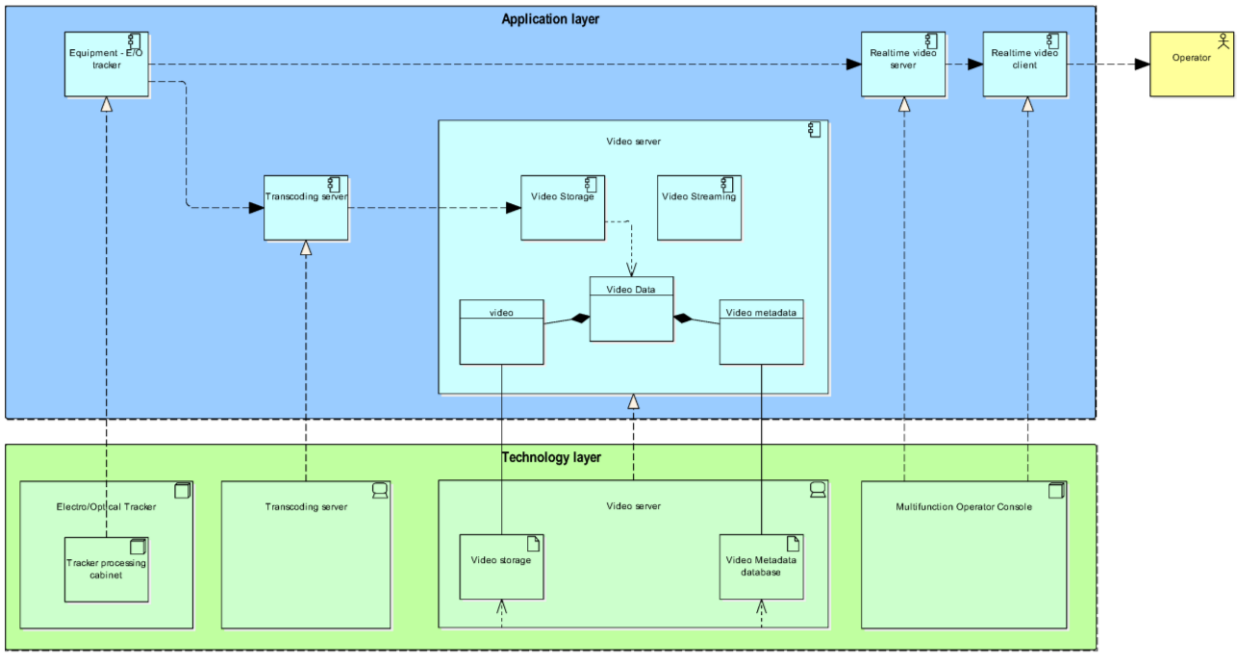


Figure 14: Application flow for real-time (low-latency) video

For some equipment, such as the Electro/Optical tracker, a video feed with low-latency is needed, for example when using the PTZ joystick control on a MOC to direct the camera to a moving target. The flow of the video data in this example can be seen in Figure 14. This video feed is done in a different (uncompressed) format and is directly sent to the MOC (which is point-to-point, unlike the multicast feed between the video streaming application on the video server and the MOC in the previous example in Figure 13) on which a real time video server application and real time client libraries run. To store these uncompressed videos the transcoding server is used to convert the uncompressed video feed into a format that can be used by the video server, on which it is then stored.

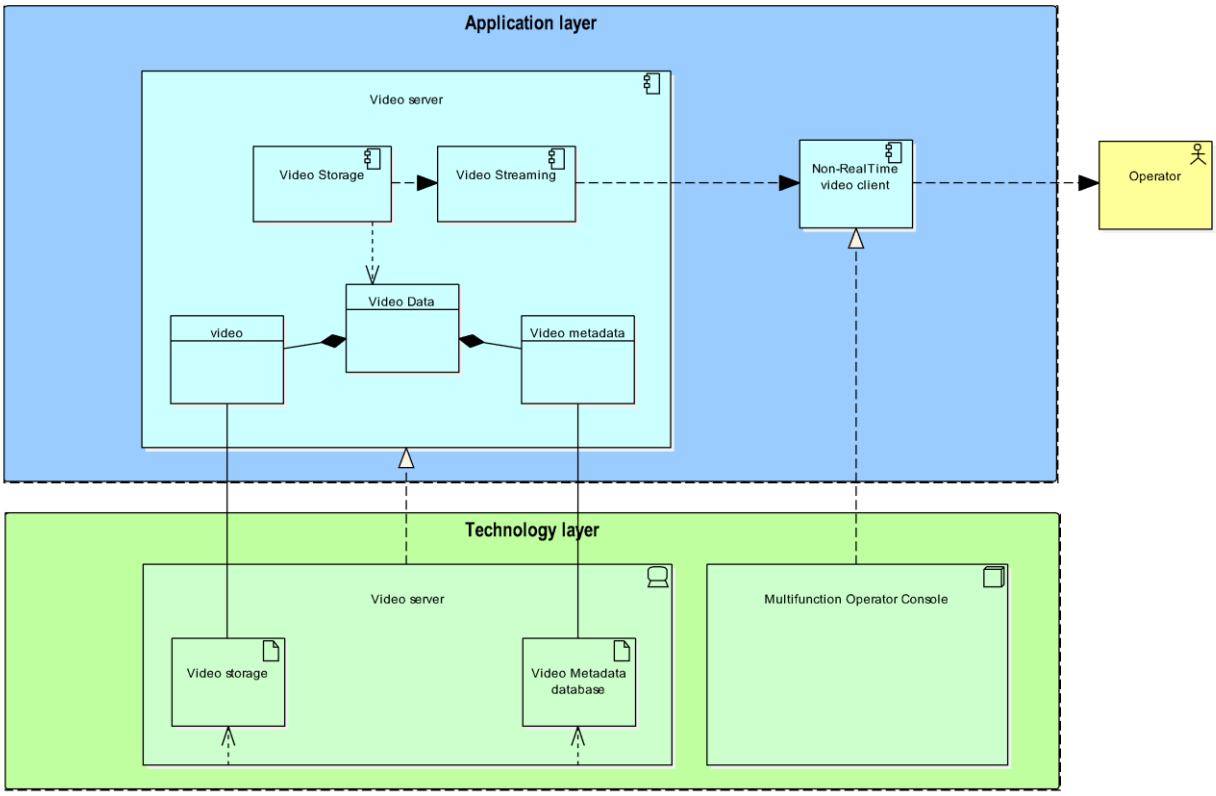


Figure 15: Application flow for video replays

In the next scenario (as can be seen in Figure 15), an operator wants to replay a stored video, and selects the relevant video (which is stored on the video server) in the non-real-time video client on the MOC, based on the video metadata that was retrieved from the video server. This application then sends the request to the video server to make a feed available. The video storage application on the video server sends this feed to the video streaming application. The video streaming application then streams this feed to the non-real-time video client on the MOC.

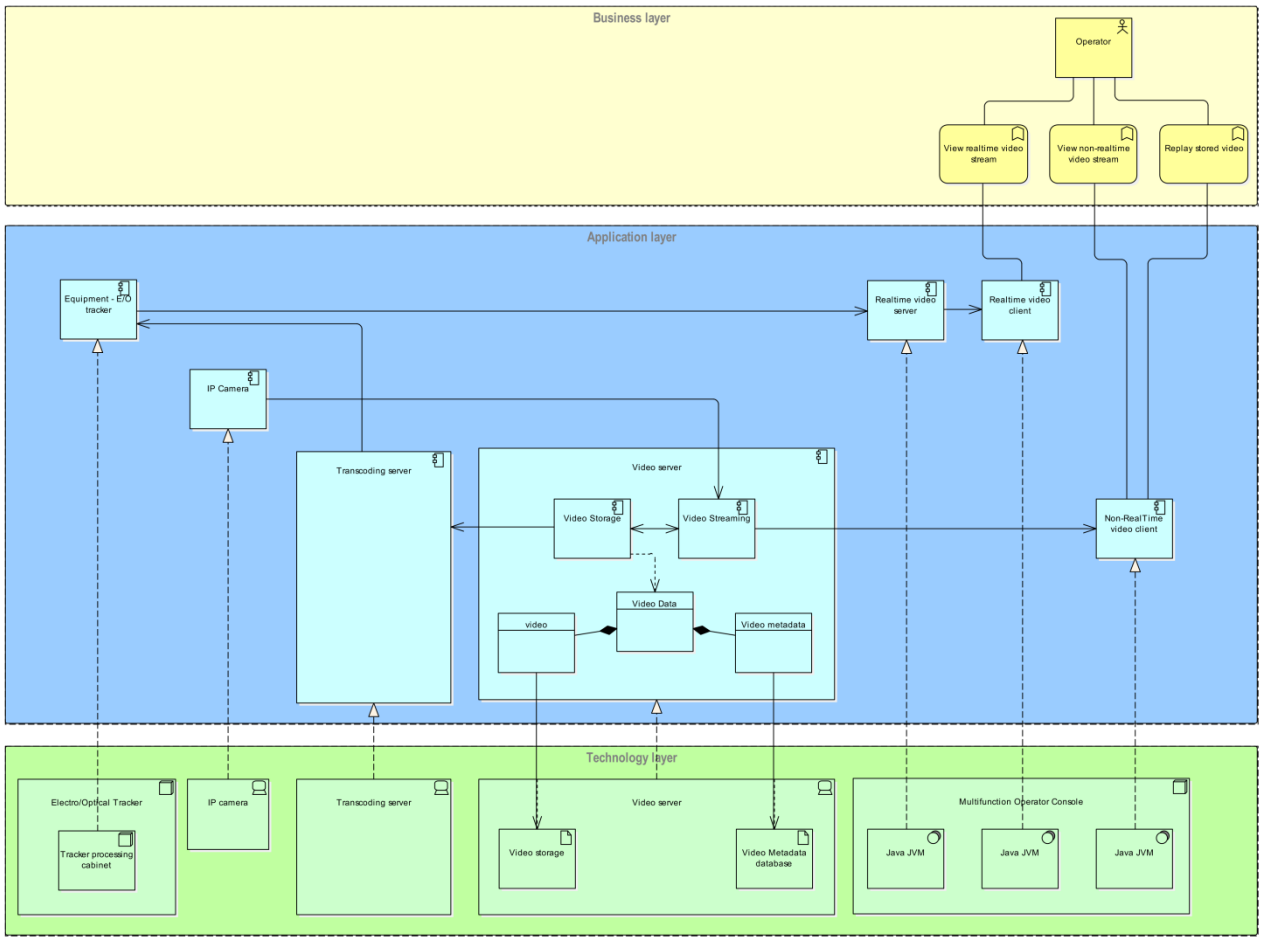


Figure 16: The overview of the applications and hardware supporting the three video flows

Figure 16 combines the systems from the previous three flow diagrams, showing the dependencies between the tasks, the applications, and the hardware. One can see that for changes in the functionality of these applications the whole chain (spread over multiple physical locations) must be changed.

3.3 Use case 3: training simulations

A large part of daily operations for crewmembers aboard a naval vessel is related to training for future situations: familiarizing oneself with the equipment, and ingraining their operational decision making and execution. Thus the TACTICOS system also has functionality to train it operators in its operation (Figure 17).

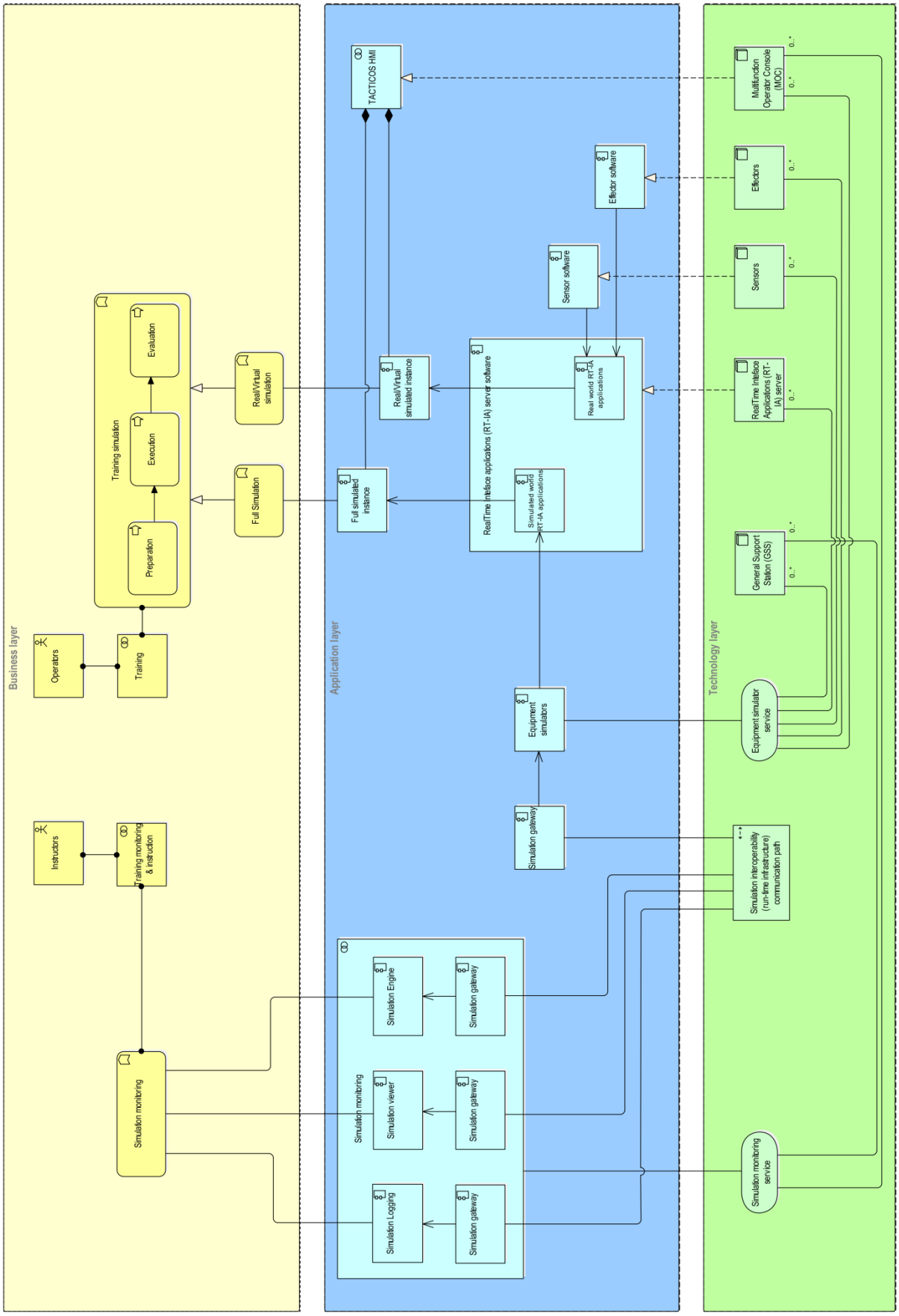


Figure 17: The architecture involved in training simulations

When the operators are training they will go through sequences of preparation -> execution -> evaluation. There are two ways to train with the system, either the full simulation mode, where all important systems are simulated, or the real/virtual simulation mode, where a combination of real sensors/effectors and virtual sensors/effectors is used. The Tacticos GUI application needs to provide instances for both training modes. The real/virtual instance will use a combination of RT-IA server Interface Applications connected to the software of the real sensors/effectors and Interface Applications connected to equipment simulators. These equipment simulators get input from proprietary simulation software as to what they should simulate.

On the other side the instructors need to monitor the simulation and instruct the operators. The proprietary software used for the simulation consists of the simulation engine that controls the entities in the simulation, a logging that stores what happens, and a viewer that gives the instructors an overview of what is happening (or happened) in the simulation. These programs communicate between themselves, and with the equipment simulators that provide simulated outputs of what is happening in the simulation, through simulation gateway software that set up an interconnected communication path for the simulation interoperability, called the Run-Time Infrastructure (RTI).

When looking at the technological infrastructure of the simulation architecture the simulation monitoring software of the instructors can run either on a GSS, or on a MOC. The operators' GUI runs on the MOCs. The RT-IA server software that performs computations and connects either to the real sensors/effectors or the equipment simulators runs on its own server. The different equipment simulators can run off of several different kinds of devices: the GSS, the RT-IA server, the sensors, the effectors, or the MOCs. The equipment simulators can be spread out amongst these locations.

Chapter 4 - Architectural Styles

While an architecture is a concrete arrangement of elements for a specific system, an architectural style is the abstraction of the characteristics that a group of architectures (with a specific purpose) have in common. The architecture of a specific system can later also be used as an architectural style, as a blueprint for another system. Architectural styles are a useful concept to reach agreement amongst architects in their preliminary understanding of the purpose and constraints for a system (Perry & Wolf, 1992). Enterprise Architecture Frameworks (EAFs) give a methodical way to organize information about an architecture. EAFs give prescriptions for views from which an architecture can be described. Different stakeholders can be more interested in one view over another. Because the information about an architecture gets structured in similar concepts, comparisons between architectures or architectural styles can also be made (Alghamdi, 2009).

4.1 Literature Review - Search & Selection Strategy

The goal for this literature review was to look for information on how an architecture for web applications could be structured, as applicable for LAN-based web applications and services. To find the different kinds of styles that can be used for the architecture a broad search is done, to present a covering view (as far as possible under the time and resource constraints of a master thesis). Starting with a search through Scopus to find the state of the art (sorted by publication date) of scientific research, filtered for journal articles and conference proceedings to find quality papers. Afterwards a search in google scholar was done to find more relevant results, due to the search engine employed in Scholar, as well as the abundance of sources therein. After the broad search to present a covering overview, a concept-centric search was done to find more specific information about the architectural styles to be able to describe and use them. This is done with Google Scholar and google search, but is not integrated into the literature review results themselves.

Search Terms
Architectural styles web development
Lan-based architecture
Web architectures
Network architectures
Web server architectures
Software architectures
Systems of systems architectures mission critical

Table 4: search terms used for the literature review

These search terms gradually emerged based on earlier literature found for the thesis as well as during the search process. These terms where then used to search through both Scopus and Google Scholar. To reduce the amount of results in Scopus the search was done with filters on document type, subject areas, and publication years until the amount was manageable. For Google Scholar the results were filtered to exclude patents, limit to publications since 2012, and use only the first 20 results per search term. See Appendix A: for the full search terms as used in the respective search engines. The results from these searches compiled the long list.

Further selection was done by looking first at the relevance of the title, then the relevance of the abstract, and based on the availability of the full text. This resulted in the middle list. Final selection and categorisation was done by skimming through the articles for relevant concepts relating to architectural styles for web development. The categories are defined by the concepts as used by the authors. Some articles featured

multiple concepts and thus appear in multiple categories. The result of this can be seen in Appendix B: This list was used to make sure no relevant concepts were missed when designing the architectural migration path.

4.2 Architectural Styles

Representational State Transfer is a style that consists of a set of principles to follow for interactions between systems (Zuzak, Budiselic, & Delac, Formal modeling of RESTful systems using finite-state machines, 2011). It contains several principles/constraints for the communication architecture: 'Identification of resources', All content is defined as a resource with each resource having a unique identifier; 'Manipulation of resources through representations', every resource can be manipulated by a standard set of manipulations (think of HTTP's GET, PUT, POST, and DELETE verbs); 'Self-descriptive messages', the messages contain all necessary information to complete a request; 'Hypermedia as the engine of application state', Information about possible future requests (e.g. page navigation) must be contained in the responses (Bohara, Mishra, & Chaudhary, 2013). The most important elements of the REST style are: URI's to individually identify the resources, and simple operations like GET, PUT, DELETE, and POST (Halili, Rufati, & Ninka, 2013). This architectural style can increase flexibility, decoupling, scalability and help distributed applications (Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012).

Services are the main components of Service Oriented Architecture. Most of Web 2.0 patterns are based on SOA (Halili, Rufati, & Ninka, 2013). Systems can be defined as services (provided functionality), that are then organised in a composition. These services then enable easier access to the functions and operations of an application (Halili, Rufati, & Ninka, 2013). The services can be seen as the basic building blocks and composing their interoperations can create powerful applications (Sheng, et al., 2014). The core principles of Service Oriented Architectures are: reusability, composability, loose coupling (scalability), Interoperability, discoverability, governance, and statelessness. Of these principles the main motivator to use Service Oriented Architectures is the increased possibility to re-use services. For stand-alone applications, that do not need integration, Service Oriented Architectures are unnecessary (Halili, Rufati, & Ninka, 2013).

Microservices are small lightweight services (a particular kind of service oriented architecture) that are decentralized by nature. When designing the services taking standards into account the underlying technologies can differ, this means that legacy code can also be re-used as long as there is an interface that is built with the common standard in mind. While designing microservices one should factor in the possibility of failure and build in resilience (Gonzalez, 2016). The size of the (micro)services should depend on the following rule: "*The microservice should be small enough to be managed and scaled up (or down) quickly without affecting the rest of the system, by a single person; and it should do only one thing.*" (Gonzalez, 2016, p. 13) A microservice should have a single responsibility, delineated by the boundaries of the correlated business process. Communication should be through calls to an exposed Application Programming Interface (API). This API needs to be technology agnostic to support loose coupling, and also creates the possibility to use different underlying technologies optimised for the desired benefits. Due to the isolated nature of microservices resilience can be strengthened by compartmentalizing failures, but the networked nature of microservices creates other resiliency problems. Scaling can be done while efficiently targeting small parts of the system, while with monoliths the whole system needs to scale at the same rate. Because of the small scale of individual microservices their deployment can be done faster. The size of microservices can be adapted to the preferred size of the development teams working on them. When the size of an individual codebase is smaller replacing it becomes easier. (Newman, 2015). Microservices have emerged from the best practices of several development paradigms: domain driven design, operation automation, development operations, the cloud, and systems integration (Thönes, 2015). Due to the distributed nature of microservices skilled developers are needed to handle the development of correlated technology, for example load balancers and service registries. (Balalaie, Heydarnoori, & Jamshidi, 2015) With a lot of small microservices one would need to prevent a large overhead,

as well as development differences between development teams, and they can create more complexity in the communication between services (Gonzalez, 2016). It is hard to deal with the complexity of microservices' distributed nature. Thought has to go into how the services deal with scaling and resiliency. Microservices are not right for every project or every company, a decision needs to be made weighing the pros and cons on a case by case basis (Newman, 2015).

Docker[4] is an open source tool that does virtualization at the application level instead of at the Operating System level, thus preventing the overhead that is associated with full virtualization. It has become very popular with its focus on deploying web applications (Boettiger, 2015). Container engines like Docker provide isolation between containers preventing cascading failures. Their low overhead, fast initialization and easy set-up lead increased development agility (Kang, Le, & Tao, 2016). With proper configuration the performance difference between native applications and applications hosted in Docker is negligible, and the same goes for network latency (Rohprimardho, 2015).

Web services enable seamless and efficient interoperability (Velasco, While, & Raju, 2013). A definition by IBM is "a new breed of Web application, and they are self-contained, self-describing, modular applications that can be published, located and invoked across the Web'" (Sheng, et al., 2014). Web services achieve machine-to-machine interaction over a network, with machine-readable interfaces ((Chinnici, Gudgin, Moreau, Schlimmer, & Weerawarana, 2003) as cited by (Traore, Kamsu-Foguem, & Tangara, 2016)). The architecture for web services typically has four main layers: The Service Transport Layer to transport the messages between services and applications (Like HTTP); XML Messaging Layer to encode the messages; The Service Description Layer to describe the way in which one should interact with the service; The Service Discovery Layer for a common repository that stores information about several services (Traore, Kamsu-Foguem, & Tangara, 2016). RESTful web services are web services that conform to the REST architectural principles (Sheng, et al., 2014). RESTful web services differ from ordinary web services in that they place emphasis on compliance to the HTTP protocol for the web service communication: Instead of a single communication endpoint, they provide (many) URI's for the different resources. Instead of a single message format like SOAP, there are many formats (like XML, JSON, HTML, etc.).Instead of the many operations of WSDL, it exposes the HTTP 'verbs' GET, PUT, POST, DELETE (Pautasso, 2014). RESTful Web Services are lightweight and stateless which makes them suitable for fast integration (Sheng, et al., 2014). The semantic web services paradigm was created for a future where one is surrounded by software agents Semantic web services are used to describe knowledge in a computer interpretable language It defines a standard to represent, publish and locate knowledge On top of the way standard web services are build up semantics are introduced so that functionality can be described by its meaning instead of just syntax Structured information can then be interpreted unambiguously (Nacer & Aissani, 2014) The combination of semantic technologies and the REST architecture can support large-scale data integration. (Lanthaler & Gütl, 2013)

With Cloud Computing virtualized services and pooled resources in a datacentre are provided to the end-user. There are three models for cloud computing services: Software as a Service (SaaS) is shared software that can be remotely accessed by users; Platform as a Service (PaaS), a virtualized environment in which applications can be deployed; Infrastructure as a Service (IaaS) deployments are virtualized resources on which an environment can be set up by the user. There are four levels of cloud deployments: Public cloud, a public model that delivers its services over the internet to the general public; Private cloud, a model where enterprises deliver the cloud services over their corporate network; Community cloud, with this model several actors pool their individual resources for joint projects; Hybrid cloud, a blend between private and public clouds (Alshaer, 2015). Cloud Computing changes the way applications on the internet are created, from the design to maintaining them (Zhou, et al., 2013).

[4] Docker software containerization platform: www.docker.com/

The principle of layered systems is well known and used (Haupt, Karastoyanova, Leymann, & Schroth, 2014) When one logically organizes an architecture into layers the coupling between the elements in those layers can be reduced ((Shaw & Garlan, 1996) as cited by (Zhang, Hansen, & Ingstrup, 2014)). Each layer takes responsibility for a part of the system functionality. A common layered architecture is the three-layered architecture where the system is split in three layers: the presentation layer, the application layer, and the Persistence layer: Presentation layer, here the user interacts with the system; Application layer, where the business logic is applied to the data; Persistence, in this layer the used information is stored (Cemus, Cerny, Matl, & Donahoo, 2016). Layering gives a greater level of abstraction, which helps in managing complexity (Zhang, Hansen, & Ingstrup, 2014). The client-server style is a well-known architectural style (Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012). The client component provides the user interface in which queries are constructed and their results eventually displayed. The server locates the queried elements, computes the results, and sends them back to the client (Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012). When a client-server architecture is used, it reduces the coupling between the components that are then deployed on the client and on the server (Santos, et al., 2015). The Internet of Things is a paradigm where objects in the physical world can perceive their environment and can be addressed over the internet ((Ashton, 2009) as cited by (Zhou, et al., 2013)). In (Zhou, et al., 2013) the Internet of Things is more broadly defined as all physical and virtual computer-embedded objects (individually identifiable by IPv6 addresses) that are interacted with through web-based services. Internet of Things devices exchange a lot of small messages, mainly real-time sensory and control data (Kovatsch, Lanter, & Shelby, 2014).

4.3 Guidelines for a new architecture

Based on the information found in the architectural styles in section 4.2 the following guidelines are proposed for use in designing new architectures that include web technology for mission- and safety-critical systems:

- In large mission- and safety-critical systems like combat management systems, with a large amount of functionality that is constantly customized and adapted, one could incorporate a Service Oriented Architecture (SOA). With it one can decouple the communication protocol between functional segments from their inner programming language, and structure functionality in building blocks that can be composed into (complex) arrangements, and can be easily re-used.
- Based on the size and expertise of the group of developers that are to develop the system a choice has to be made between a regular service oriented architecture, or a microservices architecture. Microservices divide functionality up into more but smaller services. On one hand this increases agility, re-usability and better scaling, but it also increases complexity and overhead. Thus when the need for agility, re-usability and scaling overcomes the drawbacks of increased complexity and overhead, and the developer base is up to the task, microservices can be chosen over a standard service oriented architecture.
- With large Combat Management Systems with a large amount of functionality and a need for fast integration and code-reuse, a microservices architecture could be suitable.
- For communication between web-technology-enabled entities, client-server or client-client, one could use the Representational State Transfer (REST) architectural style. This is a lightweight standardized communication protocol that is especially useful because of its flexible and decoupling nature, making upgrades of the communication protocols easier and maintenance less costly.
- To expose the services' functionality one could provide RESTful web service endpoints that because of their lightweight and stateless nature can make the integration of services fast and flexible.
- To make a microservices architecture more robust one should use containerization so each service is isolated from each preventing one from cascading their failure towards others. Containerization also

helps with decoupling the services from the underlying system software, making them more flexible in where they can be deployed.

- When a lot of microservices are deployed in a system a service registry or a reverse proxy is useful to keep knowledge of the orchestration and locations of these services relatively centralized. That way each client only needs to know the API endpoints it has to address, without having to know how the backend is set up.
- To create redundancy of the services and increase performance the reverse proxy can be combined with a load balancer, distributing the workload over multiple instances of the services, and immediately transfer connections upon failure of one of those instances.
- One should set a baseline of redundancy needed, and then try to get as close as possible within the budget of the project. Starting with multiple instances of a service on the same machine, followed by instances spread over 2 or more machines, and next up is multiple machines spread over multiple locations.
- To synchronize files between machines and increase redundancy (provided this is necessary) of the filesystem, one could make use of software defined storage, where each machine can access the same files redundantly distributed over multiple nodes.
- The web applications should be deployed to a browser that is suitable for the HTML5 functionality necessary for that web application. This can differ per web application.
- When incrementally migrating legacy functionality to a microservices architecture one can start by locating this on the systems that are upgraded, but once more and more services start to be deployed, and maintenance over a large amount of locations becomes difficult, the microservices should be centralized (with redundancy) so the amount of systems that need maintenance can be kept low.

In the next figures these guidelines have been incorporated to show how they work together. In Figure 18 one can see how a web application connects to a microservices back-end provided by a container engine running on a physical system.

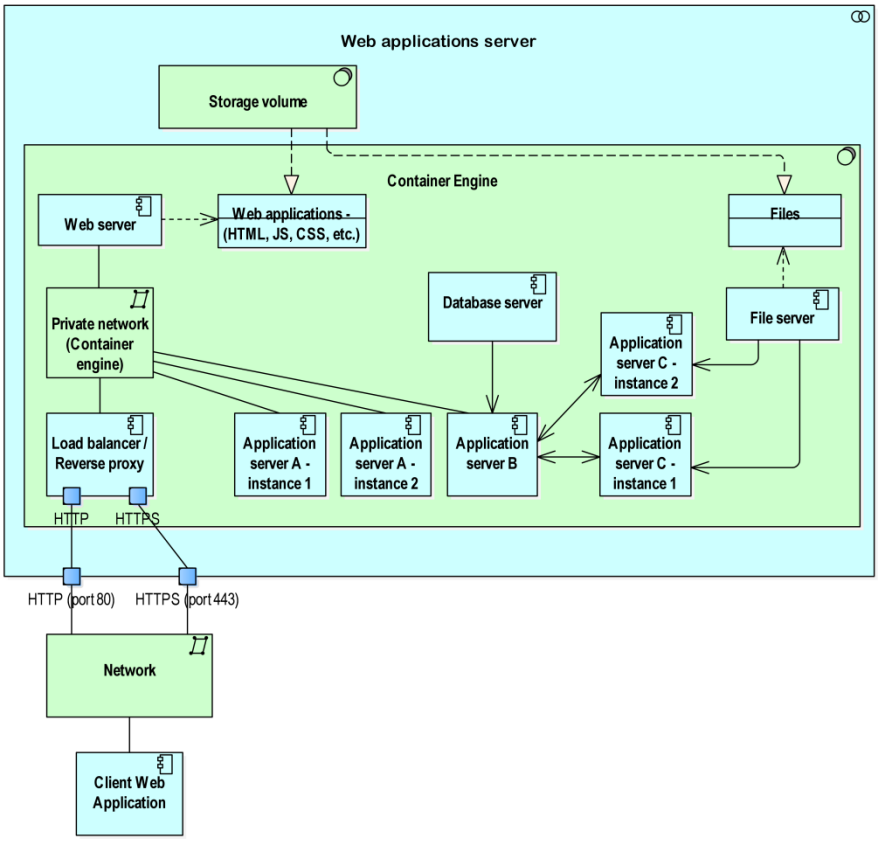


Figure 18: Containerized microservices on a web applications server

The client web application connects over the network to the server on its opened ports (in this case HTTP, port 80, and HTTPS, port 443), those ports are rerouted bij the container engine to the load balancer / reverse proxy container. This container translates the URI's used by the client to route the request to the right back-end microservices through the private/localhost network. The web client application sources are taken from the web server container, and the application data is provided by one of the application server containers. There can be multiple instances of an application server container to provide a failsafe and distribute the load. Those application servers can then connect to other microservices in the back-end, for example other application servers, database servers, file servers or other programs.

To increase redundancy and performance, one could create multiple physical servers that serve the web applications. Several measures are shown, that influence the level of redundancy and performance, in Figure 19. Although 2 physical systems are used in this example, 3 or more are also an option based on the performance and reliability needs of the system.

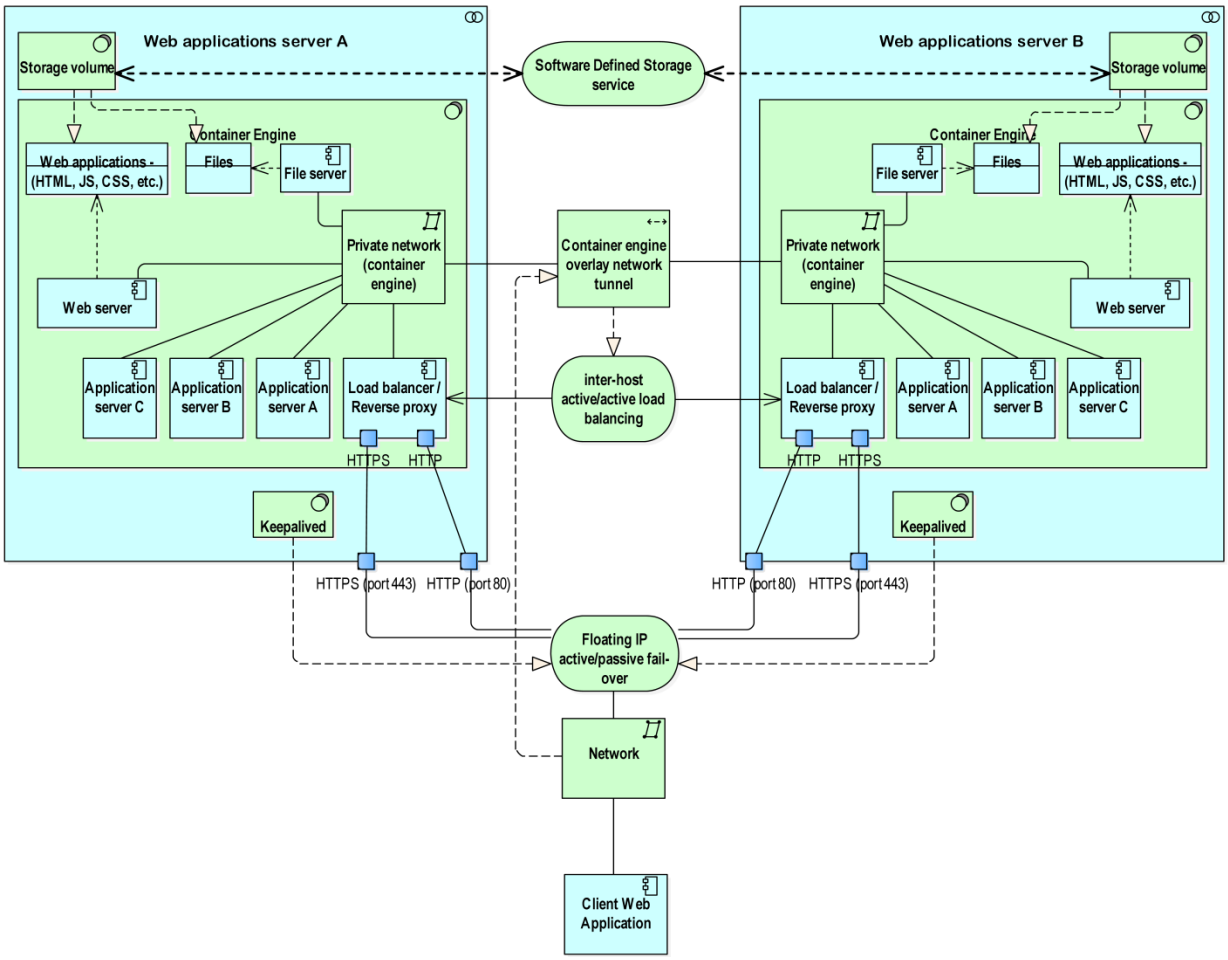


Figure 19: Redundancy measures using multiple web applications servers

A floating IP is used so the client web application only has to know one address of where the web application servers can be found. The floating IP is by default assigned to one of the servers, and all traffic will go towards that server. When the keepalived heartbeat service[5] of the other server notices the first server is down, it will claim the floating IP as its own, and after all traffic is routed to that server. Because of all traffic either going to the first or the second server, and not being distributed equal, an overlay network tunnel between the two servers' private container engine networks is set up. This makes each container on one server able to communicate to each container on the other server. The traffic is tunnelled over the physical network between the two servers. The files on both servers should be synced, especially in the case there are files that are created, modified and erased dynamically. This could be done with a software defined storage service, which syncs the local file storage volumes between servers and also creates redundancy of the files on its own nodes.

[5] Keepalived heartbeat service: www.keepalived.org

Chapter 5 - Design

Three different use cases are modelled for the new architecture, as these will cover the different kinds of systems in TACTICOS sufficiently. The first use case is the way in which a external sensor, in this case the 3D search radar, is connected to the rest of the TACTICOS system. Next is the use case of the different video streams: real time video, non-real time video, and video replays; this showcases the use case of one of the internal servers. And the last use case is the simulation architecture as this interacts with third party proprietary software. This use case also shows a gradual transition from co-existence with legacy system to a full web technology solution. An additional architectural diagram displays some differentiation in the way a sensor simulates itself (separately from the previously mentioned simulation architecture). Then the way a web applications server can be modelled with a microservices architecture is shown, for use in for example the 3D search radar, or standalone. This model is then made redundant and highly-available in the double node diagram.

5.1 Use case 1: 3D search radar

The first use case that is to be adapted for web technology in this section, is regarding the 3D search radar. This was the use case that shows how the connection between most sensors/effectors is modelled. The current situation can be seen in Figure 12. The main problem there was that in order to make changes to the application used to control the radar, the whole chain of applications between radar, interface applications server and client GUI has to be updated in each of the three locations. With different arrangements of sensors and effectors for each solution of the system, new configurations have to be made, and the same thing happens when a new version of the radar is introduced.

To solve this web technology is used to make sure that the software to control the radar is centralized server side on the radar system itself, with the client software also provided by the server. The first step towards a system with web technology is one where it operates beside legacy applications, and the applications are provided from a server in the radar system (see Figure 20). The business functions to ‘monitor status’ and ‘adapt settings’ is provided by a web application with a status component and a settings component, with the settings component also connected to the Quick Entry Keys (QEK) server software on the Multifunction Operator Console (MOC). The web application receives its data from a new application server that supplies the radar data gathered from the radar itself and the Identify Friend or Foe (IFF) system, this should be one or more containers with a REST based API in line with a microservices architecture. Next to the application server(s) a container containing web server software should be set-up, which will hold the source files (HTML, JS, CSS, etc.) for the client web application. This way all changes to source code can be made on the server side of the client-server architecture, circumventing the need to update the software on the MOC’s.

Ideally there only needs to be a connection between the applications server in the radar and the client web application on the MOC, but due to the legacy systems that will co-exist with the new technologies, there still needs to be a connection between application servers and the Interface Applications software on the Real Time – Interface Applications (RT-IA) server. The same goes between this server and the settings component of the client web application, with the same connection made to the status component through the subsystem monitor software on the MOC.

In the technology layer this will also need some adaptations, namely a new server that needs to be placed in the radar processing cabinet. This server’s performance can be adapted to the workloads of the application server programs running on it, so when relatively few programs are running on it, for example because most of the workload is still done in the legacy programs on the RT-IA server, this new web applications server can be

lightweight and thus less costly. To realize small microservices running separately (to avoid cascading failures between services) containerization software like Docker[6] should be run here, so the application servers and the web server can run in Docker containers.

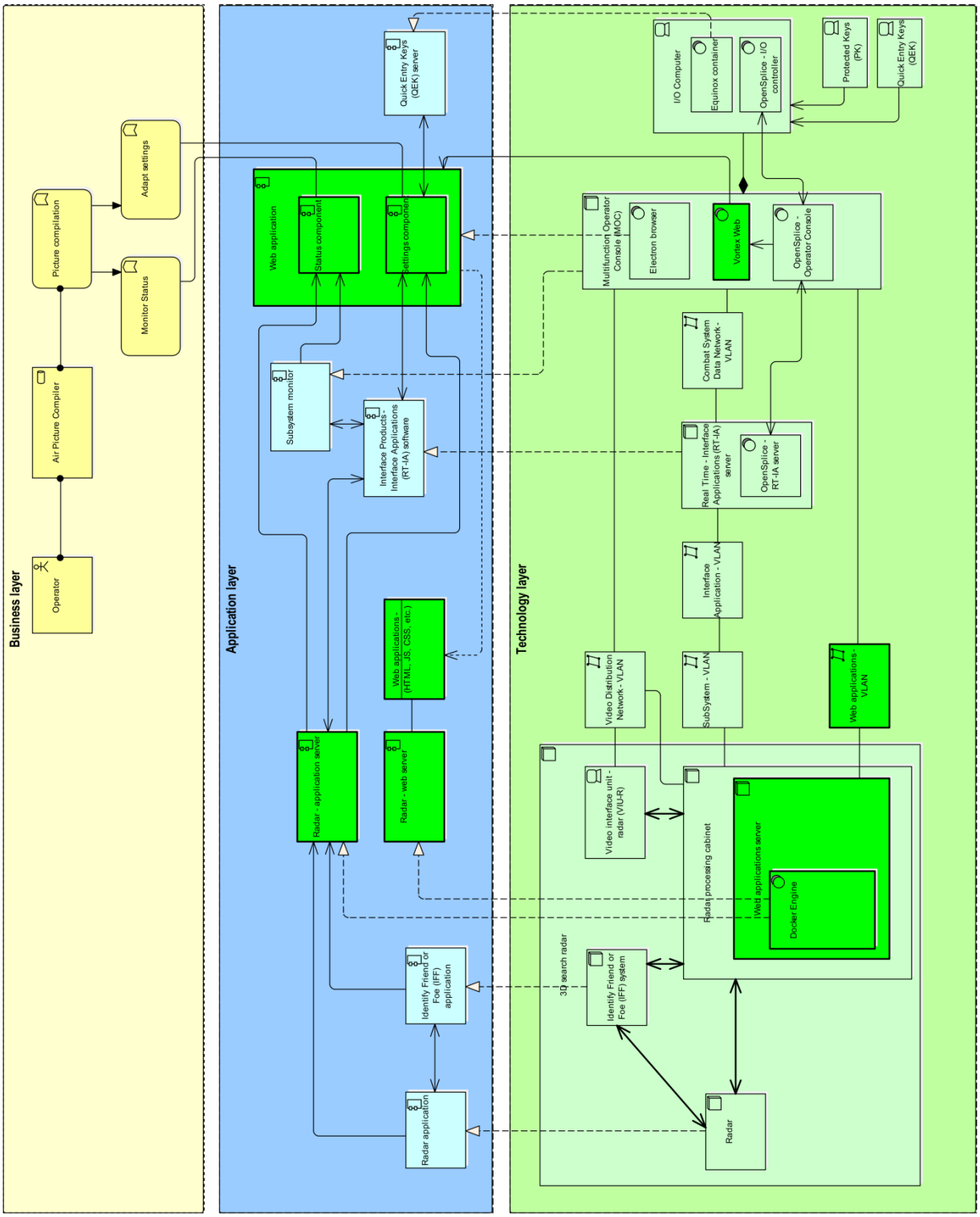


Figure 20: New web technology for the 3D search radar use case

[6] Docker software containerization platform: www.docker.com/

On the other side of the client-server divide the client web application needs to run in a browser, this can be an everyday browser like Firefox, Chrome or Microsoft Edge, but additional capabilities can be used when a browser like the Electron browser is used, that can make a web application behave like a native application.

In principle for some applications it would be enough to use the console's standard browser to connect to the web application served on an ip-address:port from the server, but that would mean one would be dependent on the browser's characteristics. Even though HTML5 is relatively mature, there is still functionality added and sometimes discarded from its specification. Which means that for some functions a more recent browser is needed[7] while for compatibility with other programs only an outdated browser version is available on the system, and in other cases one's web application can prevent this browser from being updated to a more recent version.

Therefor for this proposed architecture we have incorporated the use of a custom browser, in this case Atom Electron[8]. Atom Electron uses Chromium[9], which is the open source software that is the foundation of the Google Chrome browser, along with the Google V8 Javascript engine[10] and NodeJS[11]. It can be used to supply a customized browser for each web application, so the standard browser of the system is not needed anymore. In principle one only has to specify to which IP address and port the browser has to connect automatically to when the packaged executable (Linux, Windows or Mac) is started.

Electron can also be packaged such that one can:

- Adjust the look and feel of the browser window itself (leave out the browser menu, lock the window size, create a unified GUI look in line with the rest of TACTICOS, etc.).
- Add NodeJS modules that can make the web-application work like it's a native application (to, for example, contrary to normal browsers be able to interact extensively with the filesystem from within the web-application, etc.).
- Package a local back-up version of the web-application, that does not need to be updated in line with the real web application, but can provide graceful failover when the connection is not available.

The most important thing here is that with Electron one has precise control over the used browser version on an application by application basis, thus creating control over compatibility with HTML5 functionality. The choice whether to use the standard browser and only provide a web application on an IP address and port or package a custom browser depends on the HTML5 functionality needed to be supported by the browser for the web application, and thus needs to be revisited for each web application.

To support the direct connection between the servers of the radar and the client web application, as well as targeting these kind of connections with Quality-of-Service (QoS) rules, a new VLAN should be set up, where all of the systems that employ the web technology should be connected to. Due to the presence of the legacy systems, which transfer their real-time data through the means of the OpenSplice data distribution service, a connection between this service and the client web application needs to be made. Vortex Web[12] is a wrapper for the OpenSplice service that provides a web service interface, so the client web application can connect to this through the localhost interface, and this can also be used in the reverse direction: injecting real-time data into the data distribution service.

[7] see, for example, www.caniuse.com
[8] Atom Electron: http://electron.atom.io/
[9] Chromium browser project: www.chromium.org/Home
[10] Google V8 JavaScript engine: https://developers.google.com/v8/
[11] NodeJS JavaScript runtime: https://nodejs.org/en/
[12] Vortex Web JavaScript API for the OpenSplice DDS: www.prismtech.com/vortex/vortex-web

5.2 Use case 2: Video server

The second use case was the one regarding video (real time video streaming, non-realtime video streaming, and video replay), where like the radar use case the applications are both on the video server and on the MOC, requiring upgrades in multiple locations (see Figure 16). This is solved by putting a web server program on the video server that can supply the source content for the client web applications to the MOC. The real-time video server web application can connect to the real-time video stream of the electro/optical tracker, and supply this to the video player web application which is used by the operator to view video. This video player can also display the streams of non-real-time video streamed by the software on the video server. See Figure 21.

In the technology layer, one needs a browser (for example the Electron browser) to run the client side web applications, as was what happened in the radar use case. The web server software on the video server is hosted inside a (Docker) container. When the only change would be the addition of web server software containerization would not be necessary, but to proof the system for further upgrades where additional (application) server software needs to be added, it would be a wise decision to already set up the container environment, and put the web server software in a container.

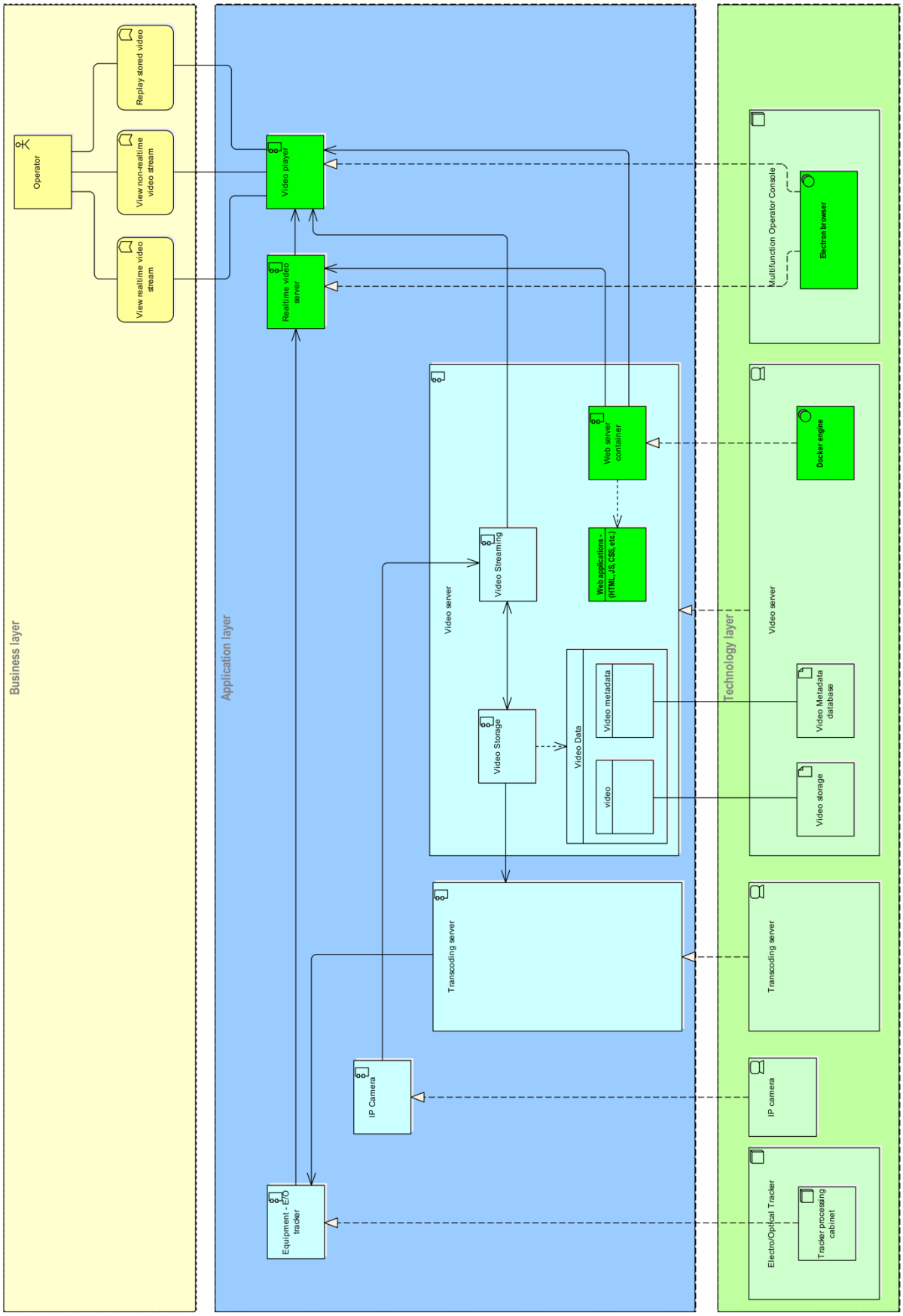


Figure 21: new web technology for the video server use case

5.3 Use case 3: Training simulations

The third use-case involved the simulation architecture (see Figure 17). The first step towards introducing web technology in the architecture surrounding trainings simulations is by introducing additional web applications for new functionality placed next to the current Tacticos GUI. See Figure 22. To support these web applications a web applications server is needed to provide the back-end for the web applications. This, like the new models for the previous use cases, contains a web server container to serve the source content for the web applications. It has application server containers to provide data for the web applications as well as doing computationally intensive workloads. Then there are the containers in which the equipment simulators will run. Due to the nature of containers they can run either legacy equipment simulators or newly developed equipment simulators, this way the equipment simulators do not need to be placed in external devices (that is, if the software for equipment simulators is available to Thales). The application servers are responsible for communication between the web applications and the equipment simulators. The application servers put the data from the equipment simulators through to the web applications and perform computational or translational steps if necessary. As long as there are still legacy programs running on the RT-IA servers that need input from the equipment simulators, the equipment simulator containers need to be able to communicate with the RT-IA server's programs. For the simulation of complex battlefields the equipment simulators need to get data regarding the simulated environment from the simulation engine, thus the web applications server needs a connection through a (containerized) simulation gateway to the simulation interoperability communication path / Run Time Infrastructure (RTI), which is the backbone of the simulation software.

Looking at the technology layer there are two changes happening here: firstly that due to the equipment simulators being included in the web applications server the 'equipment simulator service' (where an equipment simulator could be implemented in one of the following places: General Support Station (GSS), RT-IA server, a sensor, an effector, or a MOC) isn't modelled anymore. The second change is that there is now a 'web applications service' to realize the web application server system. Because the web applications server runs on an hardware-agnostic containerization engine it is flexible in where to implement this system. While it is relatively non resource intensive it can either be placed on a GSS, a RT-IA server, or a MOC. But as soon as it requires more computational resources it will need its own server hardware.

When looking at an end state of the simulation architecture (see Figure 23), where the whole Tacticos GUI is implemented as web applications, there are several changes over the previous situation: The web applications server will most likely need its own hardware, and cannot be realized as additional software on one of the existing hardware nodes anymore. As the whole simulation architecture (bar the proprietary software of the simulation engine and its related software) is implemented within the web applications server, the RT-IA doesn't need to provide the simulated-world instances of its Interface Applications software anymore, only the systems to communicate with the real-world sensors and effectors are left. The GUI web application does need to have the ability to exist as either a fully simulated instance or a combined real/virtual simulated instance (with the latter also using the RT-IA server's applications to access data from the real world).

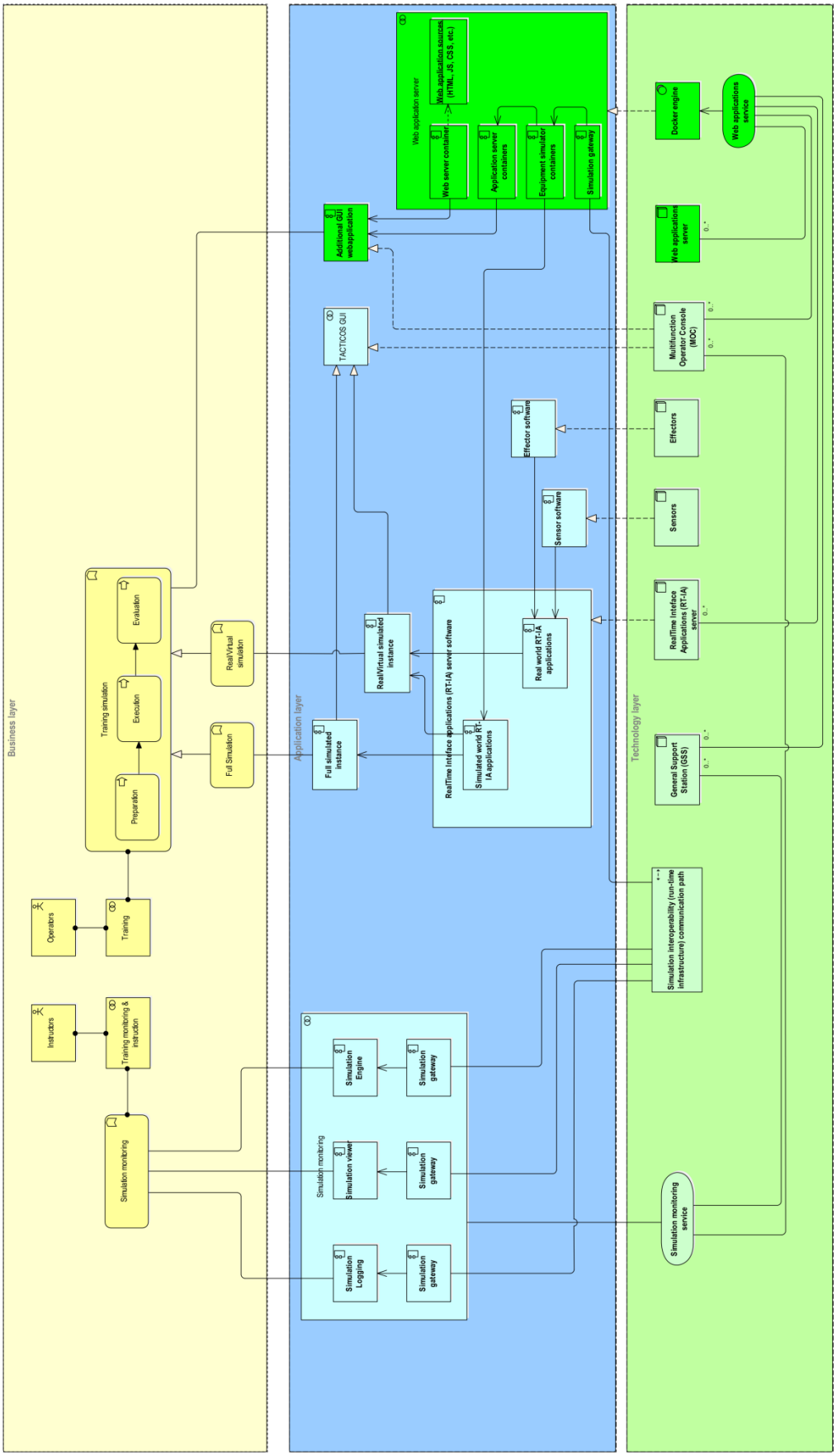


Figure 22: Additional web applications for future functionality in the simulation architecture

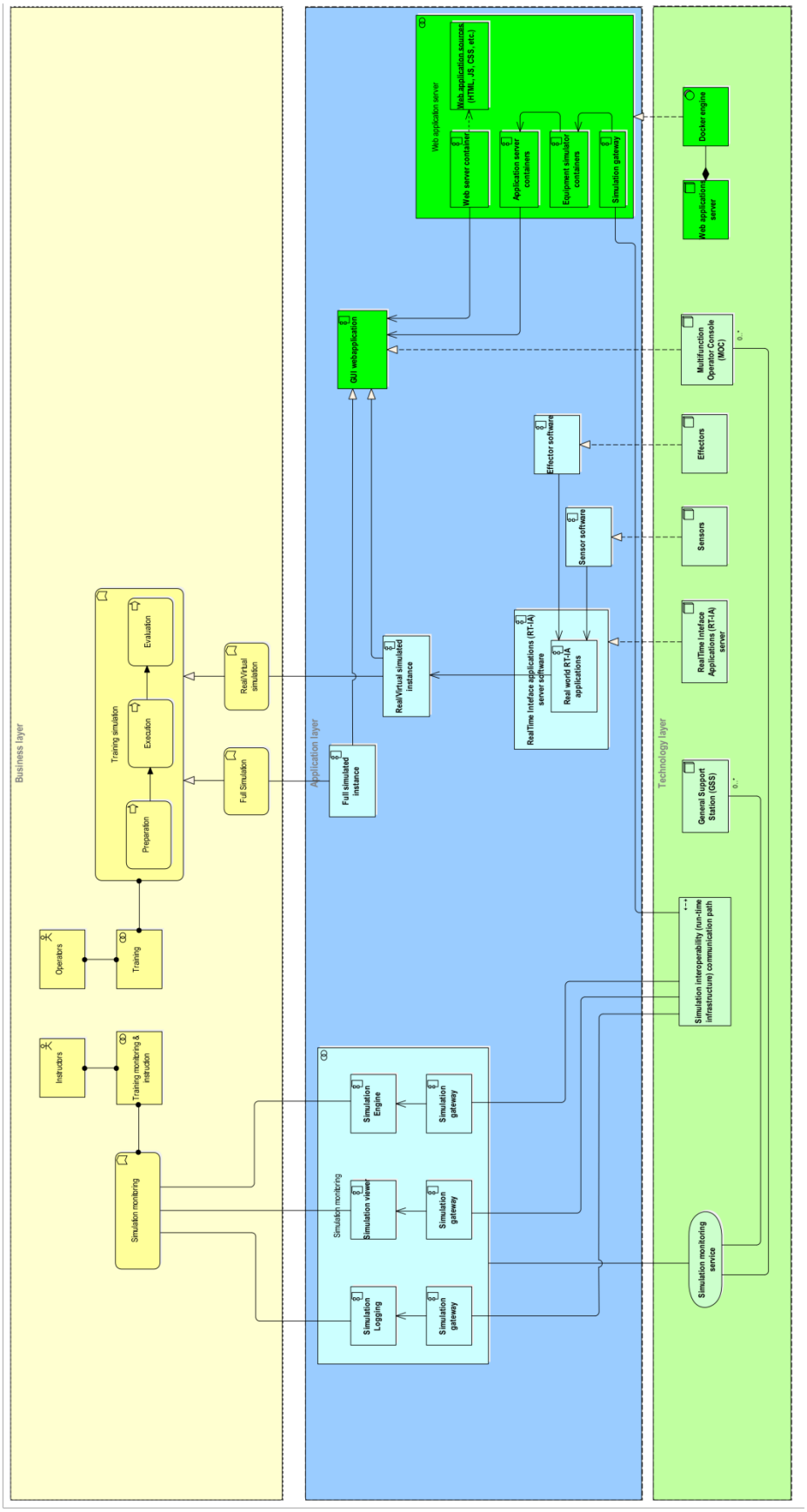


Figure 23: End state of the simulation architecture

To further demonstrate the simulation architecture the example of the first use case, the one regarding the 3D search radar, is used to model the replacement of the real radar and its software systems by their simulated versions (this is not the end-state simulation architecture, but instead the intermediate step). See Figure 24. The right side of the diagram is exactly the same as the diagram portraying the real world situation (Figure 20). The server-side application- and web-server are also the same, so communication between client and server stays the same, the only change for these containerized services is that they are now hosted on a simulation server, which in this example is realized by the GSS (as it's also already responsible for the simulation monitoring and control applications).

The real changes happen on the left side of the diagram, where the GSS also hosts the Equipment simulators for the radar hardware and the IFF system, which are connected to the simulation backbone by their simulation gateways. To avoid the need to have another version of the application server container to connect to the equipment simulators instead of the real radar/IFF hardware, a separate container that translates/redirects the communication between the application server and the equipment simulators is used.

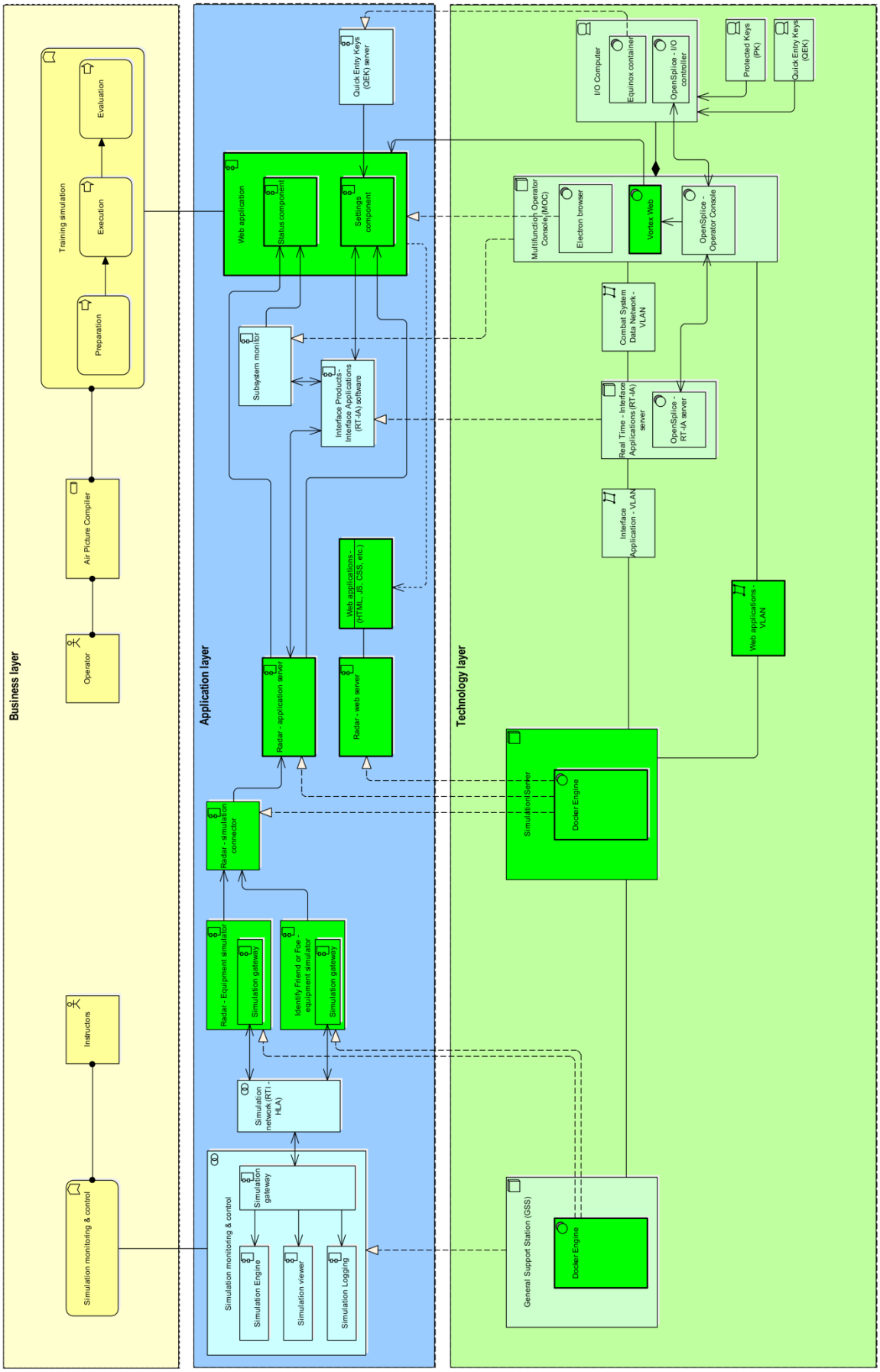


Figure 24: Simulation architecture in the 3D search radar use case

5.4 Web applications servers – internal architecture

In each of the previous use cases there was a need for a web applications server, and in its simplest form this can be just 1 or a few separate containers to which a client web application can connect directly. When instead there are more (micro) services needed to be hosted and there is a greater need for reliability/availability a more sophisticated solution is required. In Figure 25 a model for such a server system is modelled. The client web application connects through the web applications VLAN to the web applications server. It exists of several containerized (micro) services and some supporting system software. To provide connecting clients a unified/standardized HTTP gateway and API endpoint a load balancer / reverse proxy container is used (for example NGINX), this service can translate the HTTP paths and forwards requests to the right services within the private localhost/Docker network. When new or multiple instances of a microservice are started inside the service, the configuration files of the reverse proxy / load balancer can be updated to reflect the new array of services, without the need for the client application to know about it, which only needs to know about the API endpoint that it tries to use. Additional benefits of the load balancer / reverse proxy is that only its ports need to be exposed to the outside world, while the other containers stay safe within the private internal network. Also, its load balancing capabilities can immediately switch to other instances of a failing service without the client noticing.

The main services in the web applications server are the several application servers that provide computational and data serving capabilities for the client web applications. These should be small in scope in line with the microservices paradigm to prevent cascading failures by separation of concerns. With the load balancer in front of these services multiple instances of each service can be run at the same time so requests for a failed instance are immediately rerouted to other working instances, while the failed instance can be restarted. These application servers should be connected to the real-time data distribution service when needed. Therefore a (global) OpenSplice installation should be set up on the server, with a Vortex Web wrapper on top to make a web service interface of the OpenSplice data available to the application server containers. Next to these application server containers, 1 or more web server services can be set up, to provide the source contents (HTML, JS, CSS, etc.) for the client web applications. This web server component can (in the case of NGINX) also be provided by the previously mentioned load balancer / reverse proxy component, but for separation of concerns and possible throughput/performance issues it would be advisable to create a separate web server component. And for the same reason there can be a separate file server component. The contents of these web servers and file servers could be provided by a local filesystem, but in the spirit of high-availability a software defined storage solution (like Gluster[13]) could be used to provide distributed highly-available storage.

[13] Gluster scalable network filesystem: www.gluster.org

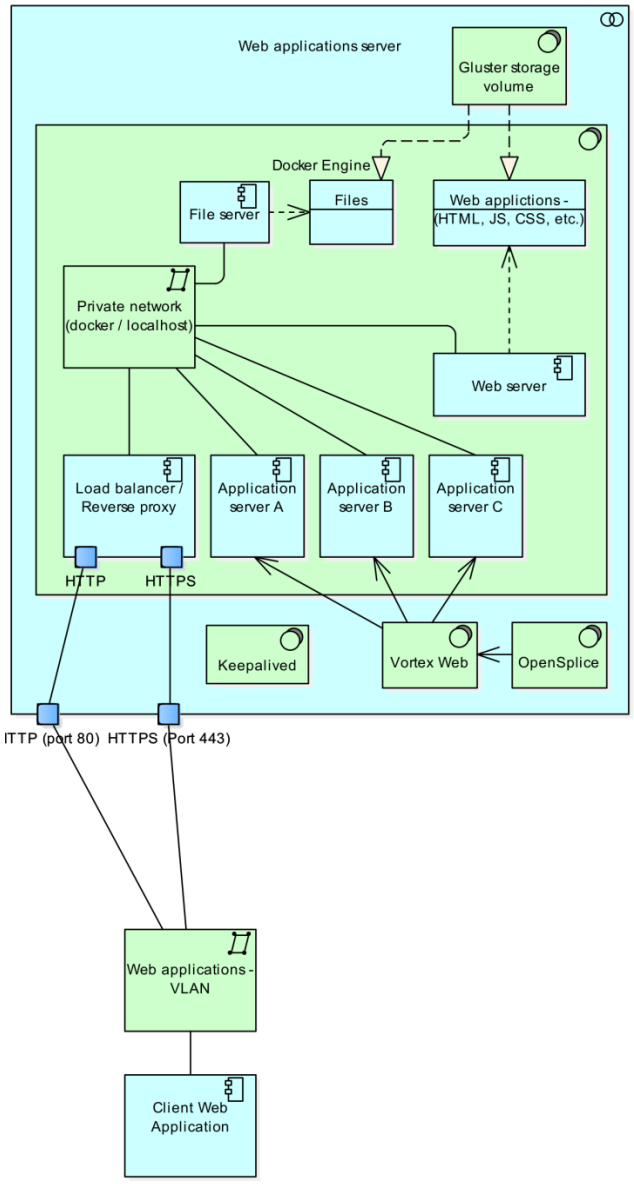


Figure 25: Web applications server - single node

In the previous model, of the single node web applications server, there were possibilities to create redundancy among its internal components, but there is still a single-point-of-failure because of the load-balancer / reverse-proxy component, and besides that a complete failure of the whole server is a risk here. When such a web applications server is used for situations where total failure is a low-impact risk, this is acceptable, but when more and more components of the total system are hosted in this web applications server, the risk can become too high. This creates the need for redundancy on the hardware server level to mitigate this risk. In Figure 26 a solution is modelled which creates that redundancy. The same node as in the single node diagram is now duplicated on a separate physical server which is placed in another location on the vessel. Ideally the web application client should not need any knowledge of the second server, because of that a floating IP service for active/passive failover, where this IP address always connects to the active server (with the other server on standby/passive mode), but when the keepalived heartbeat service[14] on the passive server notices the active server is non-responsive it claims the floating IP for itself (at the network layer), having future requests from the client web application redirected by the hardware switches to the formerly passive server.

[14] Keepalived heartbeat service: www.keepalived.org/

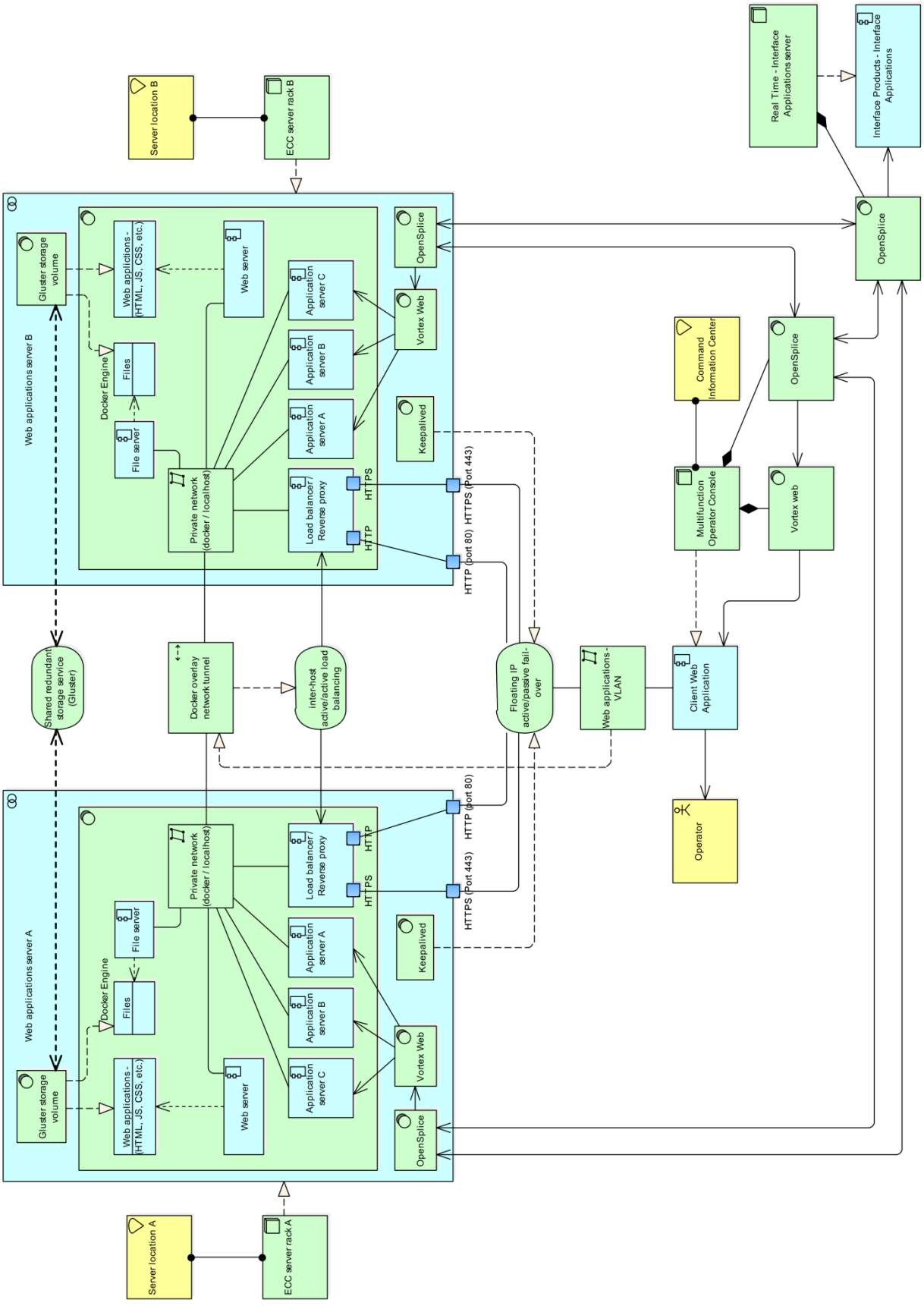


Figure 26: Central web applications servers - redundant nodes

Now the two server set-up has an active/passive nature, which means that all work is done on only one machine, to counteract this waste of performance/capacity a network connection between the two containerization engines. In this case a Docker overlay network, which creates virtual bridges on both servers, that have a communication tunnel between them over the network. Now each container on either of the servers can address, next to the other containers on the same machine, also the containers of the other server. With the configuration files of both load-balancer / reverse-proxies updated the reflect both sets of containers, the load-balancer / reverse-proxy can now redirect the requests to the containers of both servers, thus distributing the load evenly, making the most efficient use of capacity/performance.

The Gluster software defined storage solution already mentioned in the single node configuration earlier with Figure 25 can also be beneficial, next to providing redundancy for the file system, in this dual node set-up in syncing the file system between the nodes.

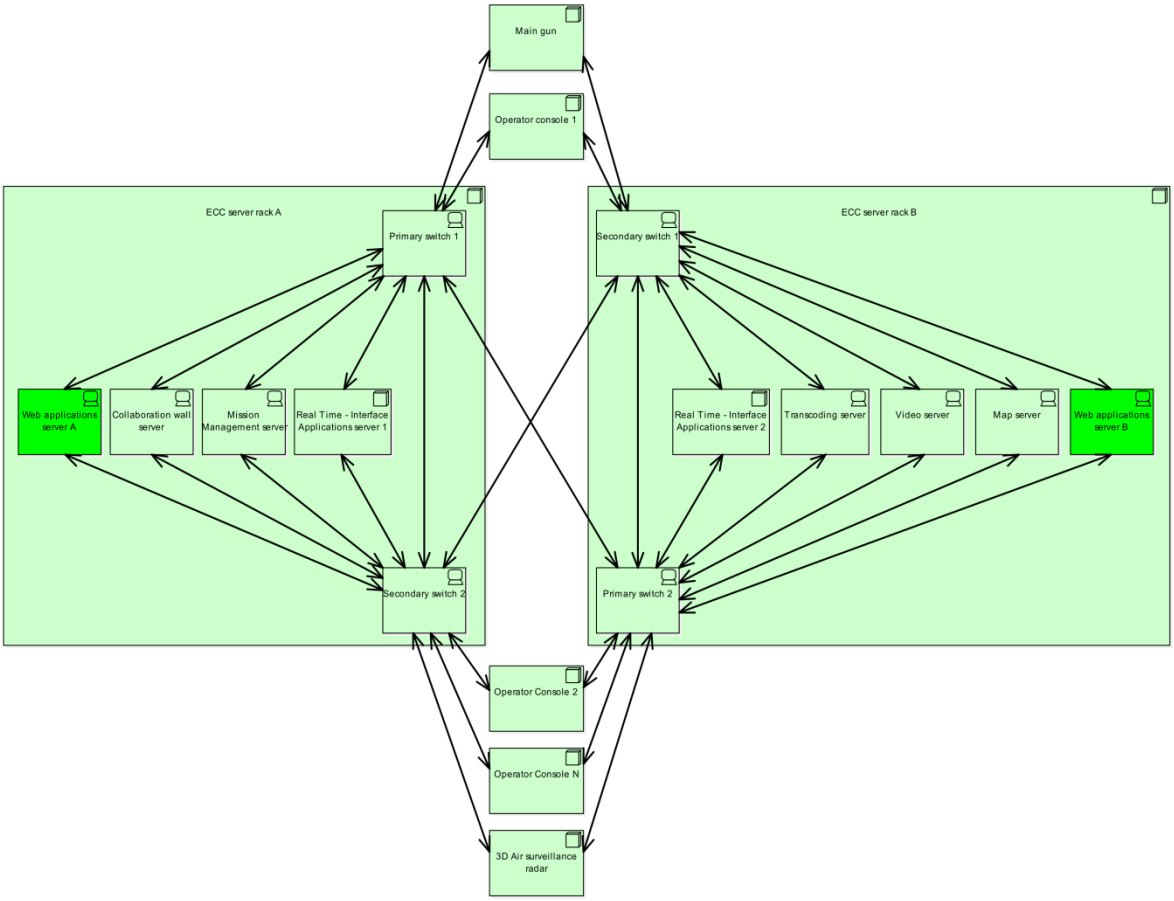


Figure 27: The dual web application servers within the context of the physical network

In Figure 27 both web applications servers (as modelled in Figure 26) can be seen added in the context of the physical network, split over the two server racks of the TACTICOS system in the vessel.

In this example a dual set-up was modelled, but this could also be expanded to even more nodes depending on the need for additional redundancy and/or performance. But the other direction is also possible, where the web application servers are not implemented on their own server hardware, but are placed on existing servers next to the existing software on those machines. The trade-offs between costs, performance and redundancy/availability of these different options need to be evaluated on a case by case basis.

Chapter 6 - Evaluation

6.1 Design evaluation

To evaluate the new architecture in its usefulness over the current architecture a questionnaire was done amongst several subject matter experts within the department. They scored the perceived usefulness of the proposed architecture over the current architecture by rating whether the new architecture better complied with each of the requirements posited in Table 3 (page 15).

They were given both Chapter 3 - ("Modelling The "As-Is" Architecture"), to unify the understanding of the current architecture among the experts, and Chapter 5 - ("Design"), containing the proposed new architecture. Both chapters were combined in a separate pdf document, along with a pdf document containing the questionnaire itself. See Appendix C: for the questionnaire. The questionnaire started with an introduction to what was expected of the questionnaire takers and the table of requirements used.

The question was the same for each requirement, it starts with the a repetition of the requirement itself, followed by the statement: "The new architecture would better support REQ-XX", and a five-point Likert type response option with neutral midpoint as originally developed by Likert (Hinkin, 1998). An additional "no opinion" option was used to prevent false responses due to lack of knowledge on the specific subject (Ryan & Garland, 1999), particularly to prevent misuse of the neutral midpoint, which is supposed to signify that there is no difference between the current and the new architecture. The response options thus are:

- No opinion
- Strongly disagree
- Disagree
- Neither agree nor disagree
- Agree
- Agree strongly

Results

The results are recorded in Table 5 below. Of the 6 people asked to participate in this questionnaire 3 gave back their answers.

The new architecture would better support REQ-XX.	No opinion	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Agree strongly
REQ-01: The system should be modular			1		1	1
REQ-02: Changes in functionality should not require the entire system to be updated.			1		2	
REQ-03: Applications should be decoupled from each other, so tasks can be done when there are resources available, and failures don't cascade to other applications.				1	1	1
REQ-04: There should be less dependency on a continuous connection between client and server				2	1	
REQ-05: The technology used for the GUI should be modern and easy to make changes to.					3	
REQ-06: Prototyping should be rapid and done often					3	
REQ-07: Processing should be done more on the server-side instead of on the client-side.				1	2	
REQ-08: There should be more of a focus on security within the network.				1	2	
REQ-09: Data should be protected from unwarranted access within the network.				1	2	
REQ-10: Expansion to mobile devices should be supported					3	
REQ-11: The system needs to be maintained for 15-20 years			1	2		
REQ-12: Applications/systems should be designed for testability.				1	2	
REQ-13: Use technology that is mainstream/common, so 3rd party components and personnel are easier to be found.					1	2
REQ-14: The deployment of a new architecture should be done incrementally and co-exist with legacy technology					3	

Table 5: results of the architecture evaluation questionnaire

The reactions, in regards to whether the requirements are better fulfilled by the new architecture, are mostly positive. Only REQ-01, REQ-02 and REQ-11 had each one vote that says that the system is not better in that regard. This pertained to modularity and updateability, where the other respondents reacted positive. And it pertained to long-term maintainability where the others neither agreed nor disagreed.

6.2 Prototype evaluation

During this research several prototypes were made to evaluate different kinds of web technology for use in the combat management system and showcase the technology to the employees in the department. In this section their goals are stated, their workings explained, and their results are discussed.

6.2.1 Sensor simulator

The first prototype was used to explore the technology of HTML5 web pages, nodeJS[15] application servers, NGINX[16] web servers and Docker containerization. Docker was used to create (and easily recreate) each of the servers in this prototype. For the 3 NGINX web servers a Docker container with a pre-installed NGINX set up was used, and for the 2 NodeJS servers a Docker container with a pre-installed NodeJS was used.

To simulate a scenario that relates vaguely to what is done in a combat management system, the simulation modelled is that of a 'track', an undefined object, that moves on a 2d map (leaflet.js[17]). This was done to keep the prototypes a bit relatable for the employees of Thales. In order to simulate a track, the track provider simulator takes starting coordinates (University of Twente campus) and a random initial heading. Then on each new interval chooses a random new heading and random distance travelled within certain parameters based on the previous heading (to prevent sudden changes in direction). The old Web Mercator coordinates on the map are then transformed by the new polar vector to designate a new location on the Web Mercator projection which is then send as a JSON message to the client which moves the track marker on the map to the new location and draws a line between the old and the new location, creating a path of previous locations.

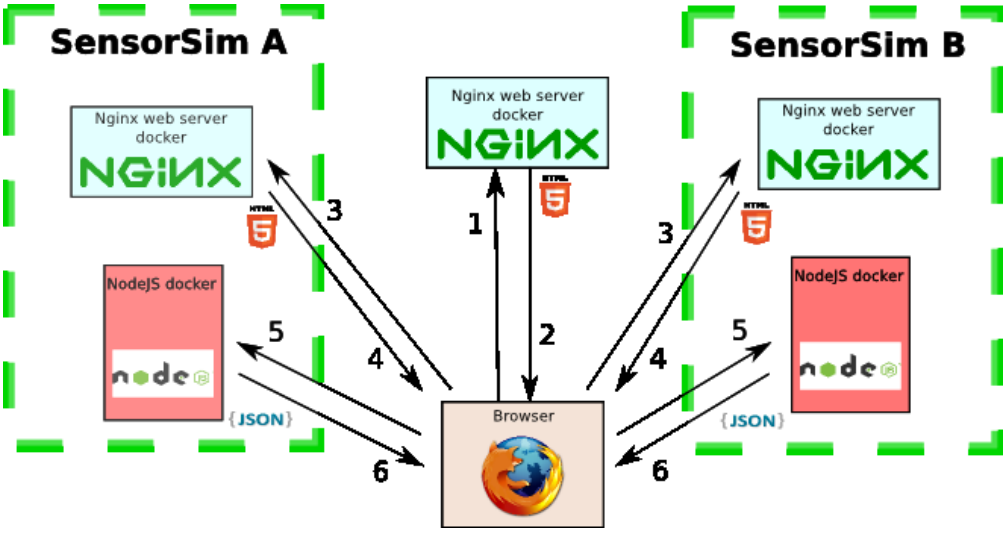


Figure 28: Architecture of demonstrator 1

As can be seen in Figure 28 there is one web server (in the middle) that serves the master page and for each of the two sensor simulators (SensorSim A and B) there is an NGINX web server container and a NodeJS application server container. In Figure 28 the numbers signify the steps in the sequence of operating the prototype:

1. The browser asks for the master page

[15] NodeJS JavaScript runtime: https://nodejs.org/en/
[16] NGINX web server and reverse proxy: www.nginx.com/
[17] Leaflet JavaScript maps library: http://leafletjs.com/

2. The server serves the master page that contains the addresses to the html-frames of the SensorSim instances. (see Figure 29)
3. The browser asks for the html-frame in the web server specific to either SensorSim A or B
4. This web server returns the html-frame that contains the data-view and interaction logic to connect to the application server. (the browser places this as iframe inside the master page
5. The browser sets up a websocket to the application server asking to simulate a track
6. The application server starts up a track simulation process and keeps sending update messages over the websocket.

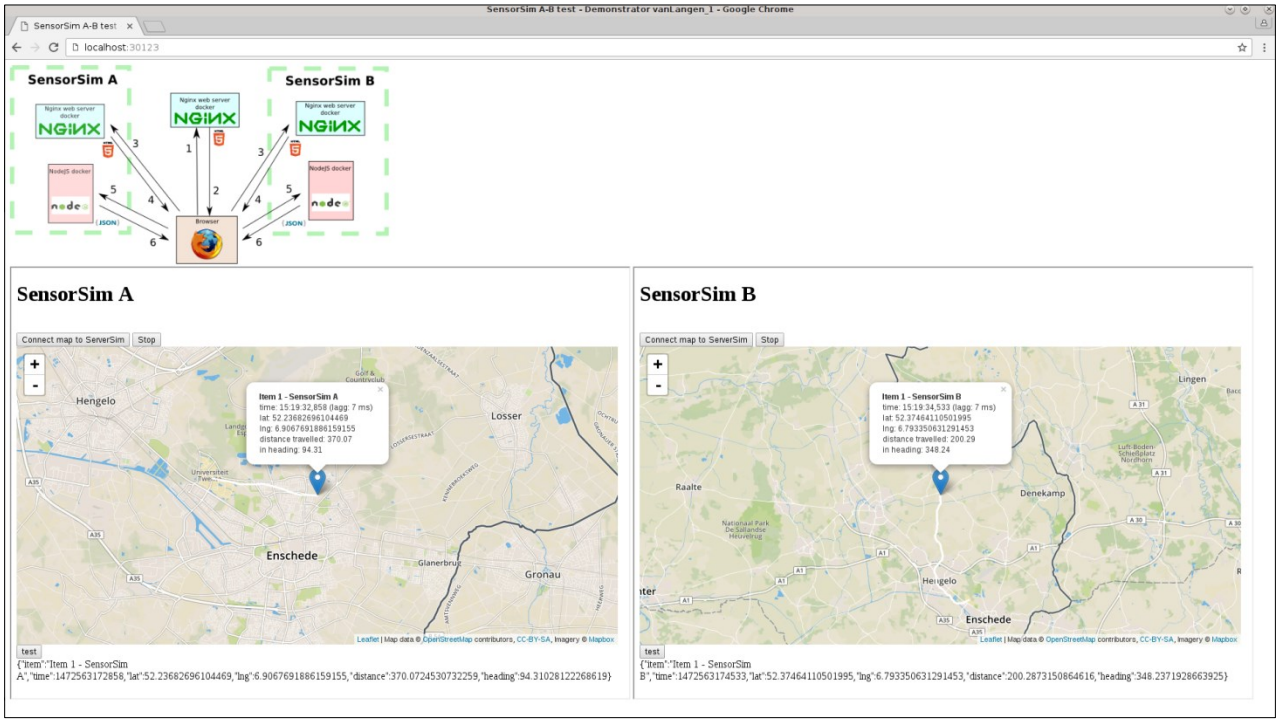


Figure 29: Screenshot of demonstrator 1

6.2.2 Publish/Subscribe service registry

The second prototype was used to explore service registry technology. When a microservices architecture is used, the amount of containers can rise quickly and it would be unwieldy to hardcode the addresses of each container in the client application. A publish/subscribe service registry can be used where the application servers can register themselves and publish their locations. The client then only needs to know the address of the service registry and query that.

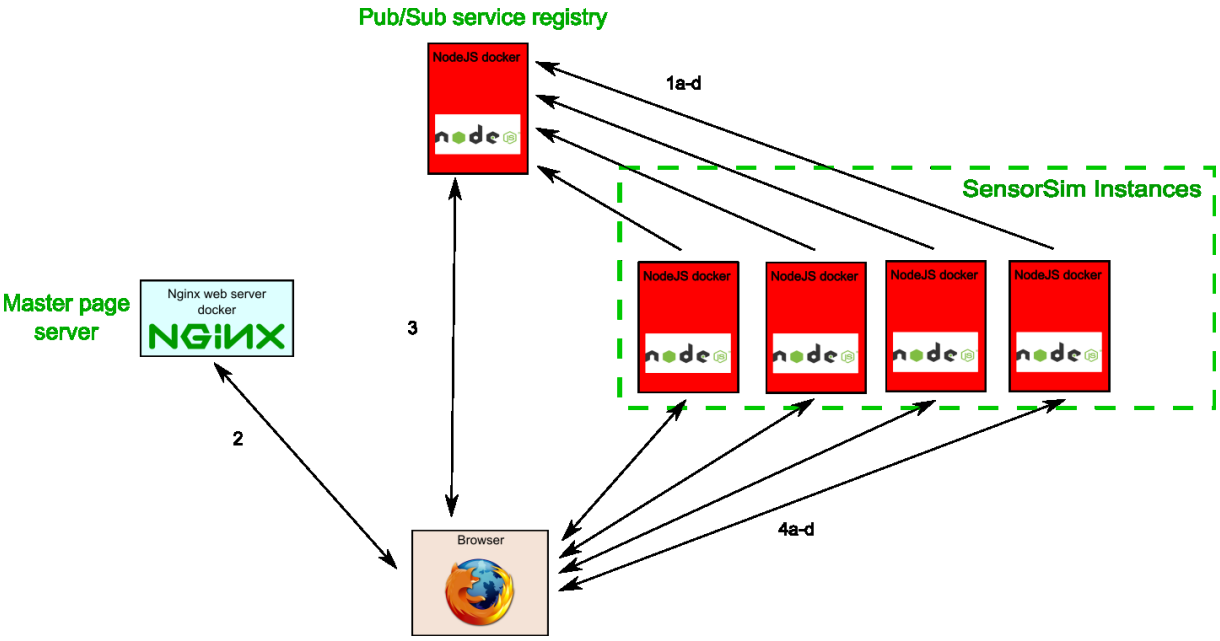


Figure 30: Architecture of demonstrator 2

In Figure 30 the architecture of the second demonstrator can be seen. There is a NGINX web server container that serves up the master page and the controller page, a NodeJS container that handles the pub/sub service registry, and several SensorSim instances (as in demonstrator 1 but with added logic to interact with the service registry).

The following sequence is shown in Figure 30:

1. a-d: The several SensorSim instances register with the service registry (after being started by the control page, for demonstration purposes, as seen in Figure 31)
2. The browser retrieves the web application from the web server (see Figure 32)
3. The web application retrieves the list of registered SensorSim instances from the service registry
4. a-d: The web application connects to the individual SensorSim instances to retrieve the html-frames which can interact with that instance through a websocket.



Figure 31: screenshot of the control page in demonstrator 2

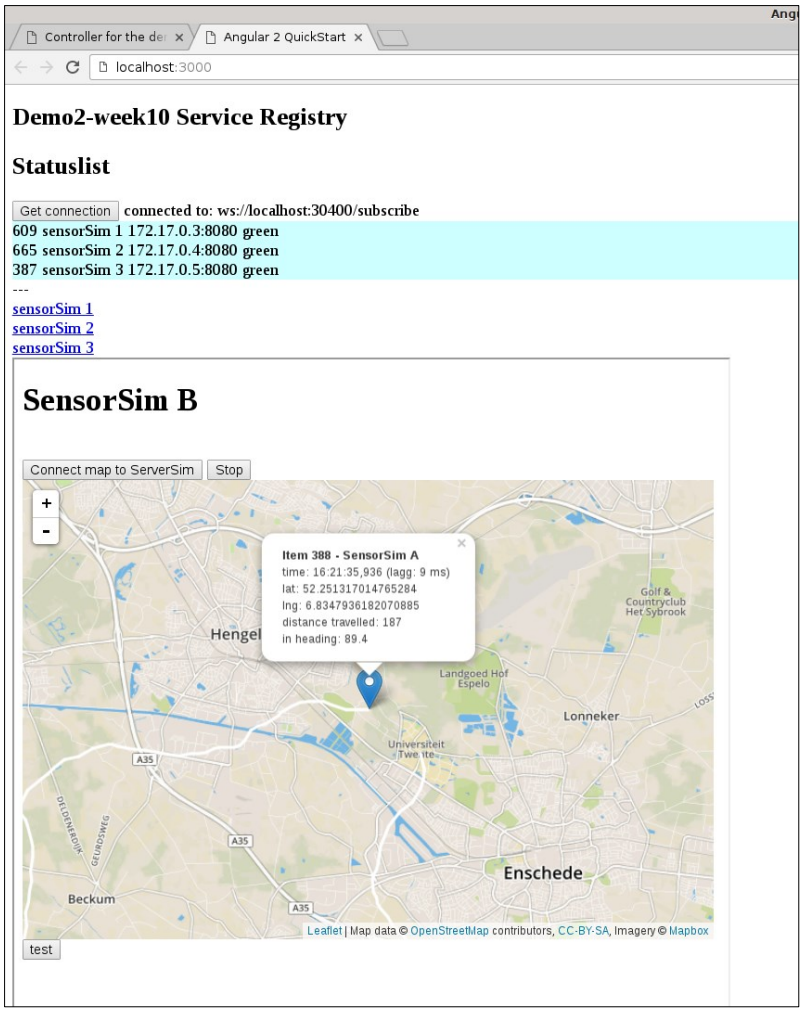


Figure 32: Screenshot of the master page in demonstrator 2

6.2.3 OpenSplice Data Distribution Service

The third demonstrator was used to explore the possibility to connect a web application to OpenSplice[18], a data distribution service. While OpenSplice itself doesn't have an interface to which a web application can connect, a separate application called Vortex Web[19] can be used as a wrapper that connects to OpenSplice and exposes a web service. When using the Vortex Web JavaScript client libraries web applications can then send and receive messages to and from OpenSplice. In this demonstrator a simple chat application was made to demonstrate this technique.

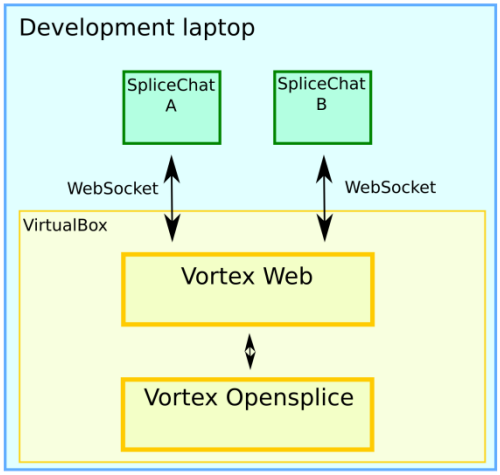


Figure 33: Architecture of demonstrator 3

As can be seen in Figure 33 a VirtualBox[20] virtual machine was set-up and both Vortex OpenSplice and Vortex Web were installed on it (this was done here because of being unable to install these applications on the development laptop, possibly due to missing dependencies). Also, a client application was started with two instances. This client application is an Angular 2 Single Page Application running in an Electron browser[21]. The client applications can then connect to the Vortex Web application on the virtual machine through web sockets and send messages through this path to each other.

[18] Vortex OpenSplice Data Distribution Service: www.prismtech.com/vortex/vortex-opensplice
[19] Vortex Web JavaScript API for the OpenSplice DDS: www.prismtech.com/vortex/vortex-web
[20] VirtualBox virtualization software: www.virtualbox.org/
[21] Electron browser: http://electron.atom.io/

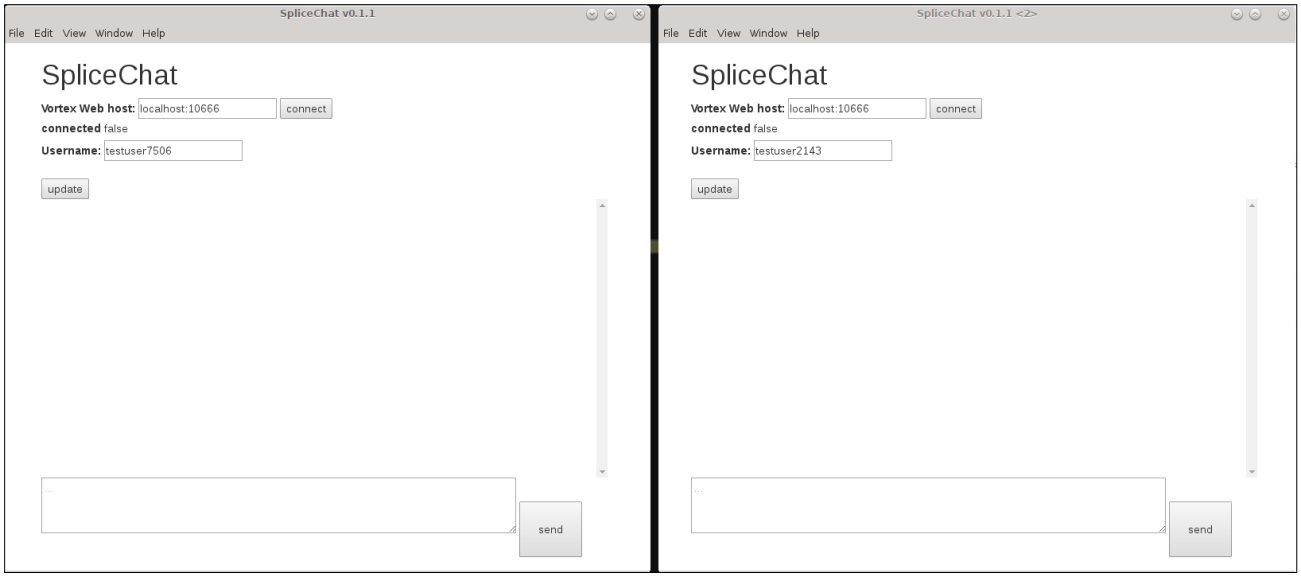


Figure 34: Screenshot of demonstrator 3

6.2.4 Custom browser with local content

This prototype was used to explore the electron browser's capabilities to serve its own content, and dynamically assign the location to retrieve a web application from. This is useful to prevent failing connections from causing the application to be unavailable. When there is a local backup web application, the application can still have functionality even when it is not as up to date as the server-side version. At the very least it can fail gracefully by notifying the user what is wrong.

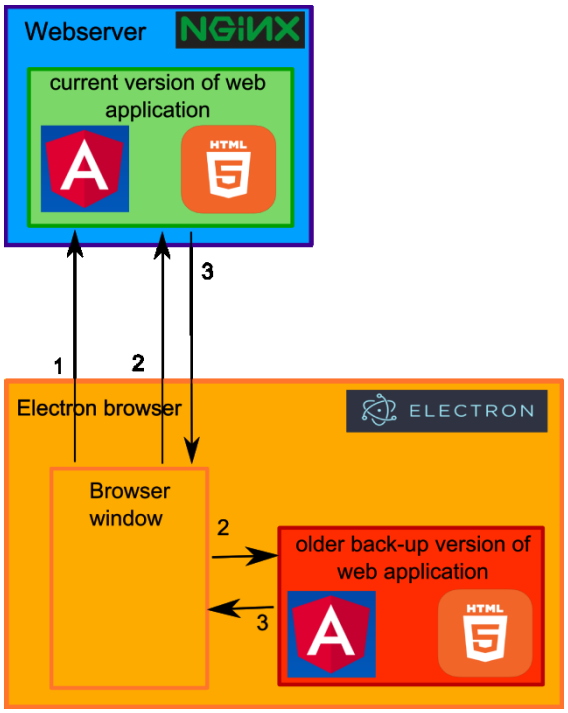


Figure 35: Architecture of demonstrator 4

The architecture of this prototype, as shown in Figure 35, consists of a NGINX web server container hosting a 'current version' of a web application (in HTML5 and Angular 2[22]) and a client browser (Electron browser) that hosts a local 'outdated backup version' of the same application. The control logic in the browser does the following steps:

1. Check whether the server is alive (in this case by trying to get the header information of a specific text file, but could also be done by a TCP lookup for less overhead)
2. In case the server is alive its content is requested, otherwise the browser local content is requested.
3. Respectively the server content (see Figure 36) or the local content (see Figure 37) is shown to the user.

[22] Angular 2 web application framework: https://angular.io/

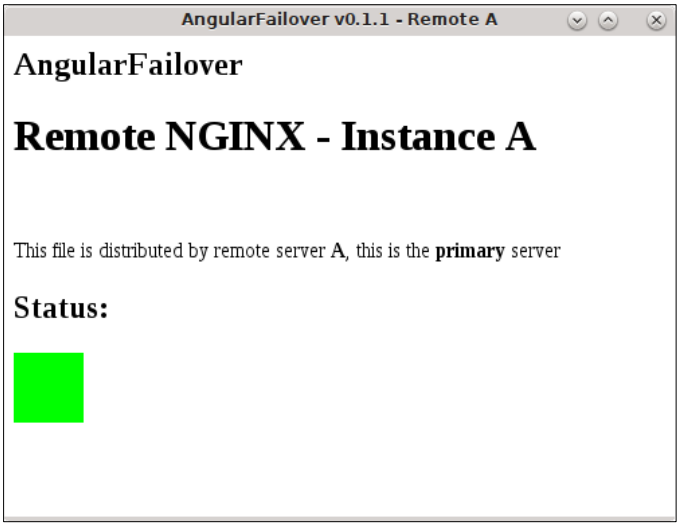


Figure 36: Screenshot of the demonstrator 4 browser window with a live server

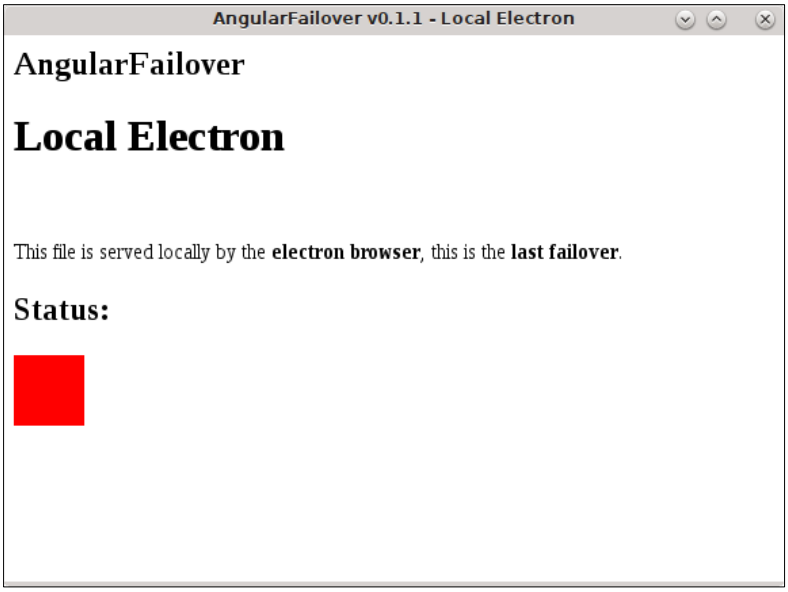


Figure 37: Screenshot of the demonstrator 4 browser window with no server to connect to

6.2.5 Containerized microservices

In this demonstrator a small microservices architecture is set up. The goal was to show how using a load balancer / reverse proxy can both make the backend process unseen by a client as well as encapsulate the back-end services in a secure and fault-tolerant environment. The client browser does not need to know more than where the web application web server is located, and the web application does not need to know more of the backend structure than the API endpoint of the function it wants to use. Also the load balancer can spread the load of multiple connections over redundant internal application servers, where a connection to a failed application server can immediately be rerouted to a different application server, without the client knowing, after which the application server instance can be restarted. The security of the system is robust as the only exposed port is the one port that connects to the load balancer (except in this case the ssh port for development and testing purposes), the internal components cannot be accessed except through the reverse proxy and thus only the physical server and the reverse proxy component need to be immunized from outside threats.

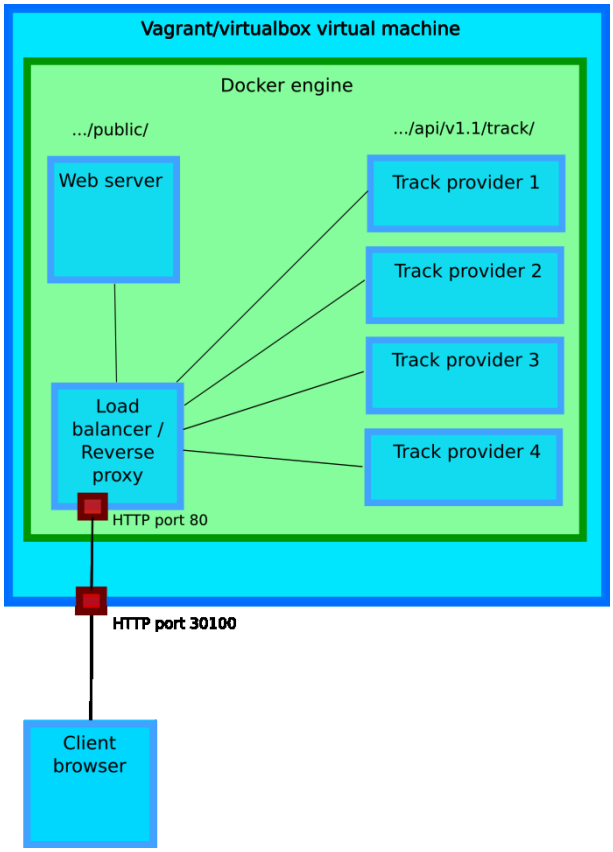


Figure 38: Architecture of demonstrator 5

In this prototype (see Figure 38) Vagrant is used to automatically set up and provision a virtual machine with Docker and one exposed port mapping to which a client browser can connect. This port is further routed to the NGINX reverse proxy / load balancer. This reverse proxy exposes the HTTP path http://localhost:30100/public/ leading to the NGINX webserver containing the HTML5&JS web application (see Figure 38) as well as the path http://localhost:30100/api/v1.1/track/ leading to any one of the 'track provider' NodeJS application servers.

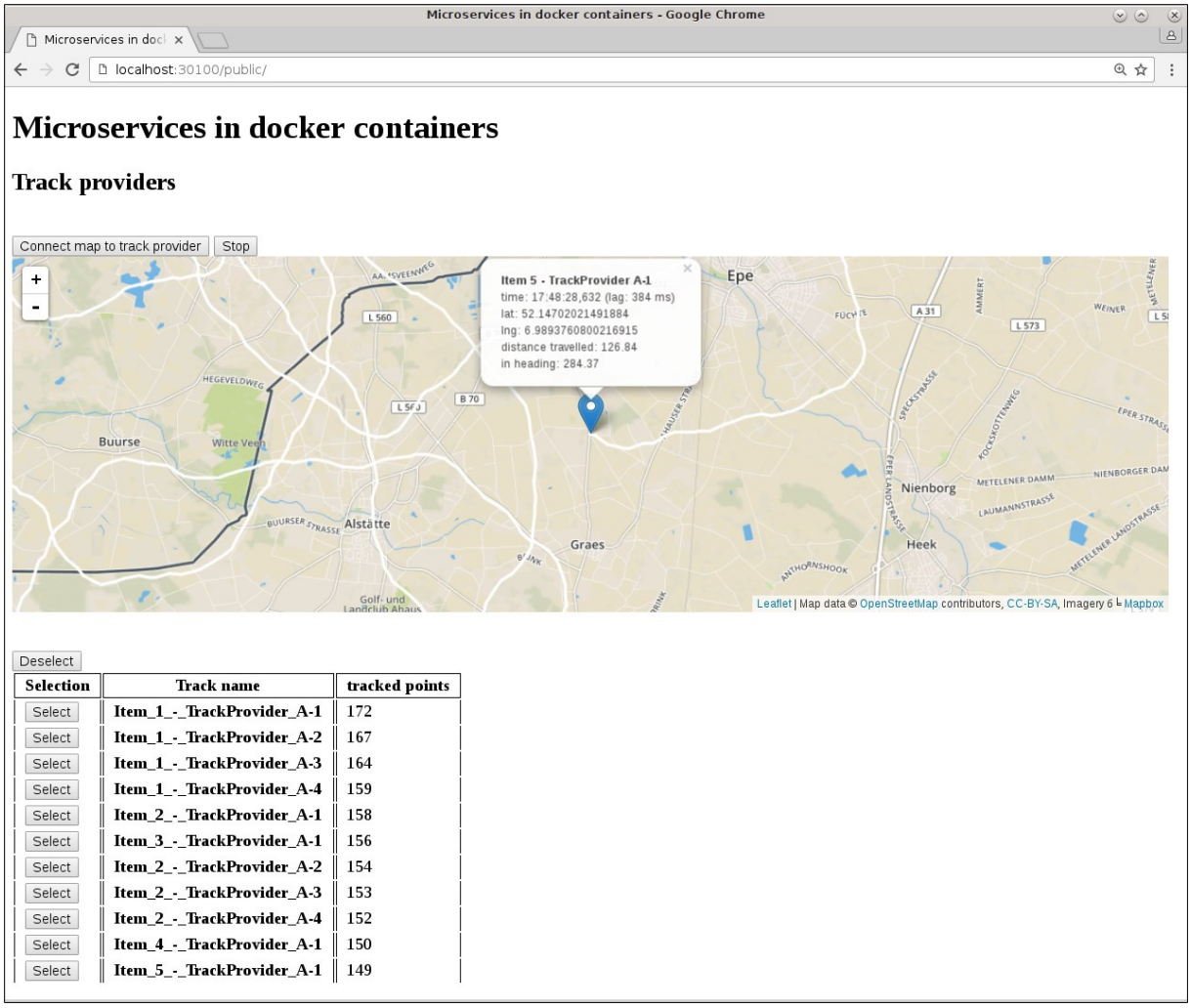


Figure 39: screenshot of demonstrator 5

Once again the SensorSim track provider and client web application from section 6.2.1 was used. The client-side and server-side code was then modified to display multiple tracks at the same time both in the map and in a new table and the origin of the track added to its data (see Figure 39), to show how unbeknownst to the client several back-end services are working to provide the data. Also when one of these services is taken down requests keep being answered (by the other track providers).

6.2.6 HTTPS/TLS security

In the last demonstrator the security, of connections between client and server, was experimented with. Both to protect the communication between client and server as well as ascertaining whether the server is who it says it is.

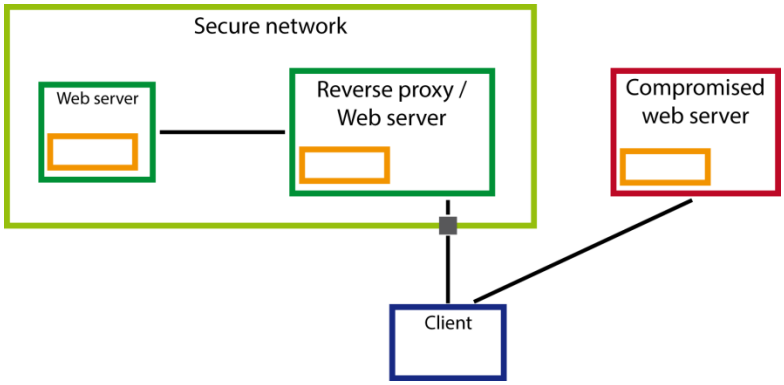


Figure 40: Architecture of demonstrator 6

The architecture of this demonstrator contains a client application, that connects to the 'reverse proxy / web server', there is a connection to a compromised server, and the reverse proxy has an unsecured connection to another web server within a secure environment (all done with Docker containers, for the purposes of testing the three server containers exposed their HTTP/HTTPS ports also directly on the localhost).

First a self-signed TLS root Certificate Authority key and certificate was made, that certificate was installed on the development laptop so it will trust that rootCA. The key was used to sign a new device key and certificate that were installed on the reverse proxy. Another 'fake' rootCA key, as well as a device key, was made and installed on the compromised server, but not on the development laptop (so the laptop will not trust this server).

The bash script to generate the (right/trusted) keys, is shown below[23]:

```
#./bin/bash
ABSOLUTE_PATH=$(cd `dirname "${BASH_SOURCE[0]}"` && pwd)

# Generate rootCA key
openssl genrsa -out $ABSOLUTE_PATH/rightkeys/rootCA-TNL-123AB.key 8192
# Sign certificate with rootCA key
openssl req -x509 -new -nodes -key $ABSOLUTE_PATH/rightkeys/rootCA-TNL-123AB.key -sha512 -
    days 20 -subj '/CN=localhost/C=NL/ST=Overijssel/L=Hengelo/O=TNL/OU=CMS/E=blah@blah.com/'
    -out $ABSOLUTE_PATH/rightkeys/rootCA-TNL-123AB.pem
# Generate device key (for on the server)
openssl genrsa -out $ABSOLUTE_PATH/rightkeys/deviceKey-TNL-123AB.key 8192
# Sign device certificate with device key
openssl req -new -key $ABSOLUTE_PATH/rightkeys/deviceKey-TNL-123AB.key -subj
    '/CN=localhost/C=NL/ST=Overijssel/L=Hengelo/O=TNL/OU=CMS/E=blah@blah.com/' -out
    $ABSOLUTE_PATH/rightkeys/deviceKey-TNL-123AB.csr
# Sign device certificate with rootCA key
openssl x509 -req -in $ABSOLUTE_PATH/rightkeys/deviceKey-TNL-123AB.csr -CA
    $ABSOLUTE_PATH/rightkeys/rootCA-TNL-123AB.pem -CAkey $ABSOLUTE_PATH/rightkeys/rootCA-
    TNL-123AB.key -CAcreateserial -out $ABSOLUTE_PATH/rightkeys/deviceKey-TNL-123AB.crt -
    days 20 -sha512
# copy rootCA certificate to development laptop rootCA cert store
cp $ABSOLUTE_PATH/rightkeys/rootCA-TNL-123AB.pem /etc/pki/ca-trust/source/anchors/
# update development laptop's rootCA cert store
update-ca-trust extract
```

[23] *an unnecessarily high amount of bits for the keys and the hashes has been chosen, respectively 8192-bit and 512-bit where 2048-bit and 256-bit are considered best practice currently (while writing this thesis), to see if any performance problems occur when using a higher degree of security.*

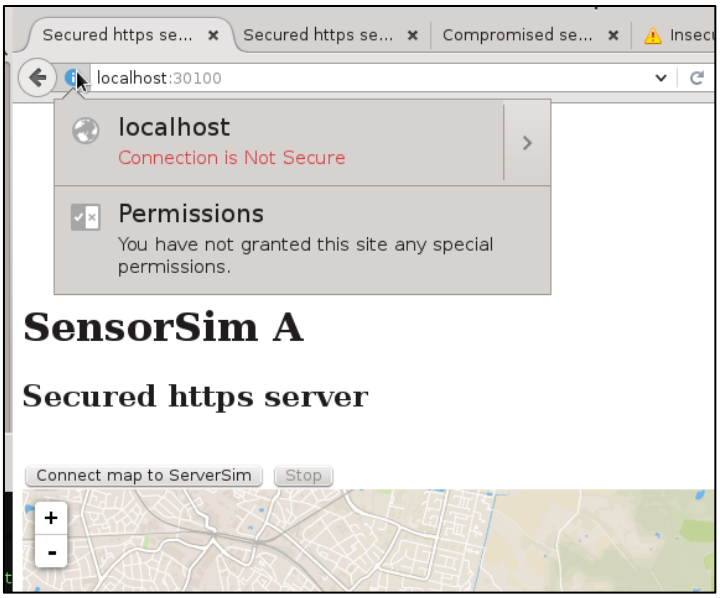


Figure 41: non-secure connection to secure SensorSim-A

When connecting to the reverse proxy over the unsecured HTTP connection the browser displays that the connection is not secured.

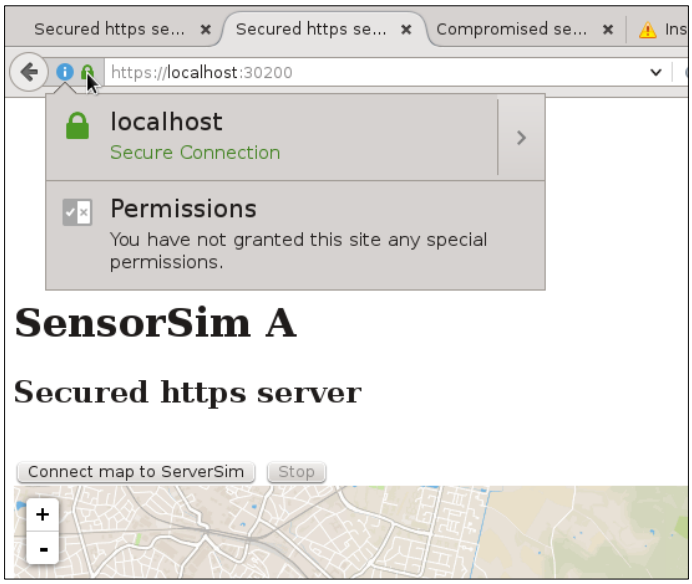


Figure 42: secure connection to secure SensorSim-A

When connecting securely over HTTPS/TLS the browser shows this to be the case.

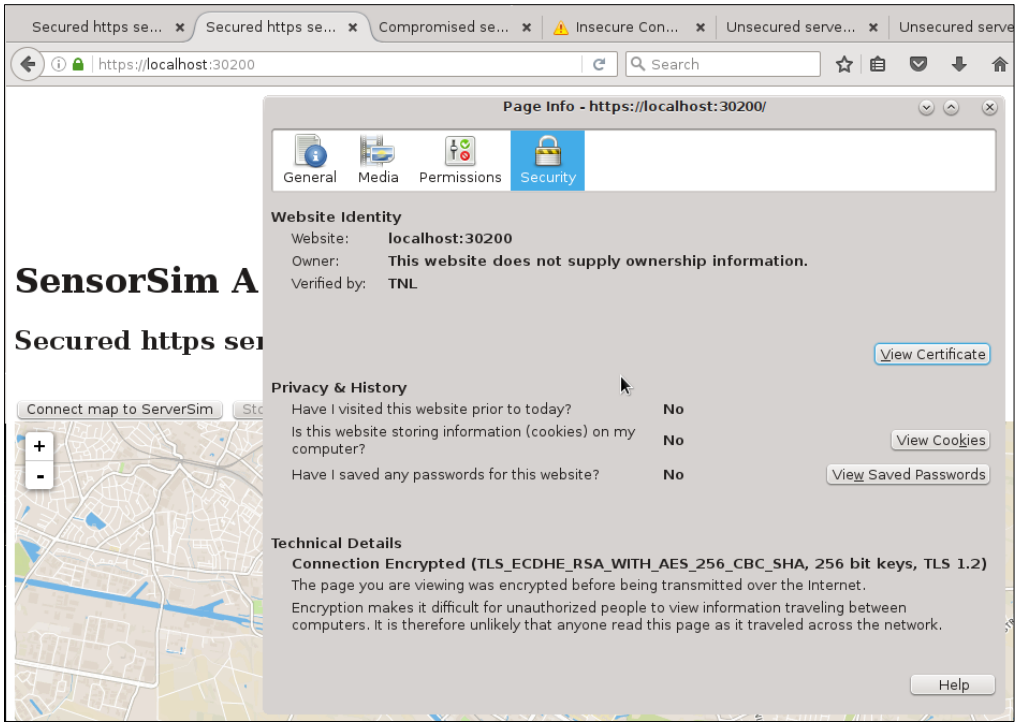


Figure 43: Details of secure connection to SensorSim-A

Here the details of the secure connection can be seen, encryption has been applied to the connection[24], and the server certificate matches with the root certificate the browser knows (see 'Verified by: TNL').

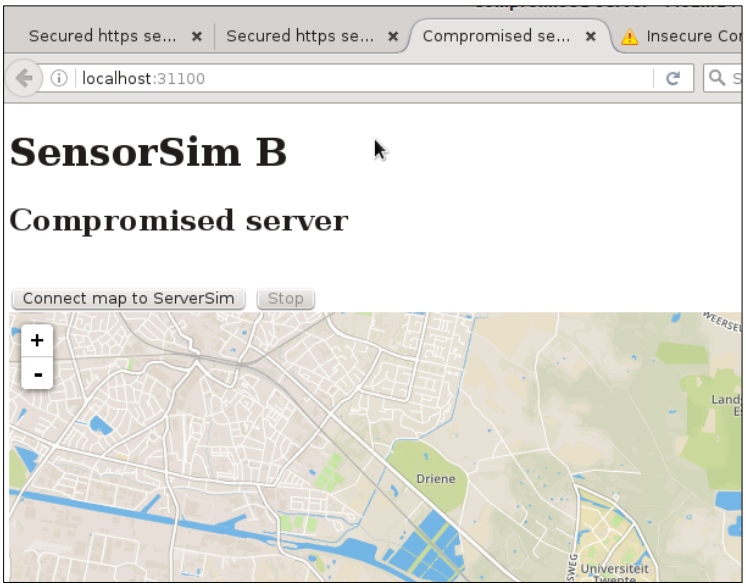


Figure 44: non-secure connection to compromised SensorSim-B

[24] Transport Layer Security (TLS) 1.2 – Elliptic Curve Diffie Helman Ephemeral (ECDHE) key agreement scheme with Rivest-Shamir-Adleman (RSA) authentication encryption algorithm - Cipher Block Chaining (CBC) of 256-bit Advanced Encryption Standard (AES) encryption for the connection with 256-bit Secure Hash Algorithm (SHA) hashes for authenticity checks.

When somehow a threat like a compromised server shows up on the network and has stolen the IP-address to redirect that traffic to itself, an unsecure connection through HTTP would not be able to know that the server is different, in Figure 44 nothing out of the ordinary can be seen.

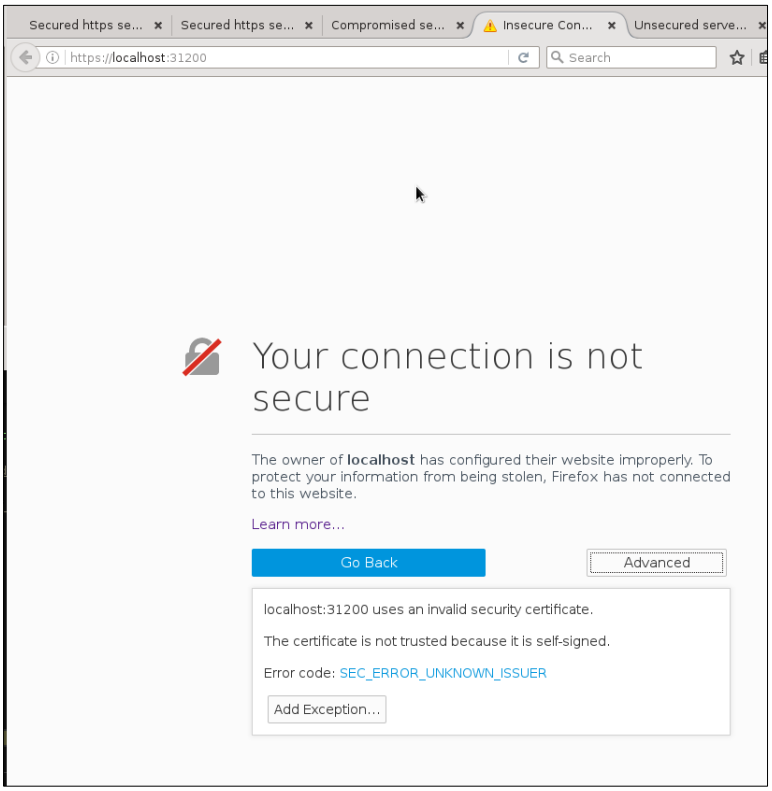


Figure 45: secure connection to compromised SensorSim-B

When one tries to connect to the HTTPS port of that compromised server a warning will pop up, as this server does not have a TLS certificate that is trusted by the development laptop. Thus it is paramount that the browsers for the client web applications connect to an HTTPS:// address by default.

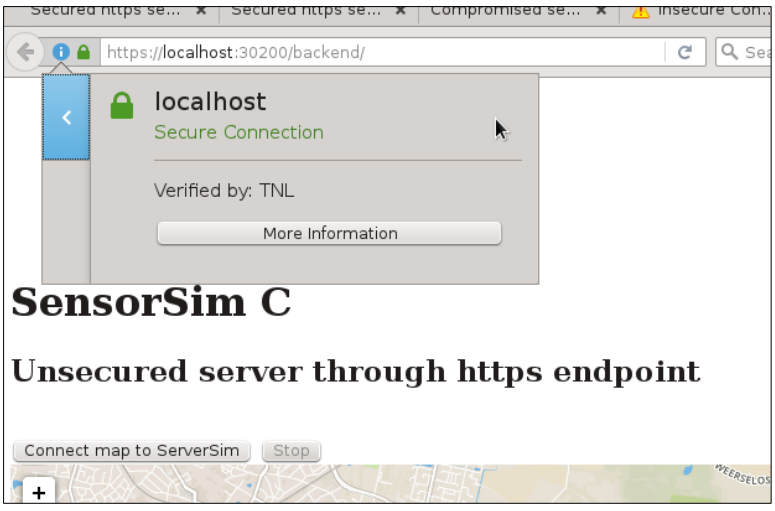


Figure 46: Secure connection to SensorSim-C through HTTPS-endpoint

In this last scenario, there is a web server in the 'secure network' back-end that does not have the software routines or the TLS certificates to set up a secure connection. To connect to it securely the client can connect to the reverse proxy, which can operate as a HTTPS endpoint, taking over the HTTPS/TLS responsibilities. This means there is now a secure connection between client and reverse proxy. The back-end connection between reverse proxy and back-end web server is not secured, but this is okay as it is inside the secure back-end network.

Chapter 7 - Conclusions & Recommendations

7.1 Conclusions

7.1.1 Research Questions

The goal of this research was:

"Design a new architecture and an architectural migration strategy for a combat management system, to exploit the benefits of web technology."

This was decomposed into the following research questions. A short conclusion on each question is given:

RQ1: *What limitations are encountered in the current architecture?*

There were several limitations encountered in the current architecture, as the result of expert interviews within the company suggest, which could be either solved or lessened when using web technology. These limitations were described in section 2.3.

RQ2: *What are the requirements/constraints for a new architecture in a CMS by Thales?*

Several requirements were aggregated from the results of the expert interviews, partly based on what the limitations of the current architecture are and how it could be changed for the better, and partly based on new possibilities that could be unlocked by web technology.

RQ3: *What kind of architectural styles are suitable when using web applications and services in a CMS?*

A literature search into web technology used or researched in recent literature turned up several concepts of which particularly web services, Microservices, containerization, private clouds and liquid software seemed promising for application in a Combat Management System.

RQ4: *What risks to the product quality could a new architecture encounter?*

Next to obvious risk categories as security, where for example the not-compiled nature of web applications means that client-side code is more easily accessible than compiled code, another category of risks is reliability related. Most web technology is made for use on the World Wide Web and thus is designed for large numbers of concurrent users and having room for occasional hiccups, wherein with mission- and safety-critical systems there doesn't need to be support for a large user base, but long term reliability and immediate response is critically important.

RQ5: *What should a new CMS architecture look like to support web applications and services?*

The steps for a migration towards an architecture with web technology were given and modelled. Even though this could only be done within a limited scope, the use cases presented herein were chosen in consultation with Thales to cover the different possible systems sufficiently.

RQ6: *How should the (long-term) migration path towards the new architecture be structured?*

Due to the modular nature of web technology and the standardized way of communicating through RESTful web services, web technology can be helpful to modernize lots of legacy systems by tackling each upgrade one at a time. Thus easing the transition towards a new architecture through greater control over costs, development time, and developer knowledge.

RQ7: *Is the proposed architecture suitable for a CMS?*

The results of the evaluation questionnaire indicated that the respondents mostly felt that the new architecture better fulfils the requirements than the current architecture. The only requirement that did not get a net positive pertained to the long-term maintainability, which is due to the volatile lifecycle of web technology.

7.1.2 Main Conclusion

Modern web technology can thus be helpful to modernize mission- and safety critical systems as the technology for use on the web and its architectures are designed to maximise uptime. With its modular nature a gradual path towards a new architecture can be taken, preventing a sudden and expensive shift towards a new architecture from being necessary. Web technology can help overcome some of the limitations of TACTICOS and unlock new possibilities.

7.1.3 Limitations

This research had to stay relatively small in scope in relation to the size of the whole TACTICOS system and had to be restricted to several use cases that can be deemed representative for most of the TACTICOS system, but do not cover it fully. This means that a different solution could be needed when the scale of the migration becomes too large for what this research investigated. Due to the nature of combat management systems information about competitors' systems is not available, and thus the proposed architectural migration could not be evaluated against other similar systems, and thus its validity as a generic solution could not be ascertained.

7.2 Recommendations

7.2.1 Immediately relevant recommendations for Thales

Thales needs to collect more information on the performance of web technology under a realistic load in a realistic test set-up, similar to what could be expected under production loads, to be able to make decisions on what parts of the system can be migrated and what kind of performance characteristics the architecture and hardware need to have. Can a web application user interface cope with what is requested of it? Does communication between web services increase network load? Does the shift of computational workload from clients to centralized servers create an overload on those servers? To gather this information an experimental prototype of web services server-side and web applications client-side with realistic functionality, realistic workloads, and a realistic infrastructure should be implemented and tested. The parts of the Tacticos architecture that are less dependent on latency, performance and reliability might be more suited for a move to web technology than parts of the system that are more dependent on those issues. Realistic load testing can provide the necessary information for the decision of which parts of the TACTICOS system can be re-done as web applications.

7.2.2 Recommendations relevant in the short term for Thales

The next step is to implement a pilot project that can be used to increase knowledge of the technology and find out what its limitations are. It needs to be small in scope so it stays manageable. It needs to have enough functionality requirements and technological requirements to be useful as pilot towards further projects, but not too much as that will create a too steep barrier. An appropriate amount of work hours needs to be allocated for familiarization with the technology, as web technology has quite a few different technological aspects to it when creating a high-performance highly-available solution, but the employees involved in this project can later use the knowledge gained to spread it among other employees for use in future projects. And there has to be room for error, as mistakes are likely to be made when implementing new unfamiliar technology.

7.2.3 Recommendations relevant in the long-term for Thales

When familiarity with web technology has increased and more is known about the risks and limitations involved, decisions can be made based on qualified information about whether to implement web technology on a system-by-system basis, and when to do this. The first priority on the way toward an architecture using web technology would be to create an infrastructure that is capable of supporting it, then develop standardized way to expose functionality as web services that can be re-used for further upgrades, after which the client-side and server-side code can be developed into web technology. The beauty of standards-conforming web services and the modularity of microservices is that the chunks that are to be upgraded are bite-size and can be handled in a manageable manner. Also some systems might be better off staying in their legacy form, but get a wrapper to expose their functionality as a web service.

7.2.4 Future work

As most uses of web technology are meant for situations with a large amount of short-lived low-workload connections, most research on performance and reliability of web technology is targeted in this direction. There are still gaps in the knowledge on what the performance and reliability characteristics of web technology become in situations with a small amount of long-lived and high-workload connections as is the case in this kind of mission- and safety-critical system.

7.2.5 Business practice

Web technology with microservices and containerization seems a promising option for situations where large legacy system monoliths, with a lot of different functionality, need to be upgraded to modern standards. This technology is suitable for fast incremental upgrades that make a migration more manageable and cost-effective. Recent advances in web technology have made it a more viable option in regards to performance than before. In Section 4.3 several guidelines were proposed for use in designing architectures with web technology for mission- & safety-critical systems, and for Combat Management Systems in particular.

References

Abolfazli, S., Sanaei, Z., Gani, A., Xia, F., & Yang, L. T. (2014). Rich mobile applications: genesis, taxonomy, and open issues. *Journal of Network and Computer Applications, 40*(1), 345-362. doi:10.1016/j.jnca.2013.09.009

AbuJarour, M., & Awad, A. (2014). Web Services and Business Processes: A Round Trip. In A. Bouguettaya, Q. Sheng, & F. Daniel, *Web Services Foundations* (pp. 3-29). Springer New York. doi:10.1007/978-1-4614-7518-7_1

Aerts, A. T., Goossenaerts, J. B., Hammer, D. K., & Wortmann, J. C. (2004). Architectures in context: on the evolution of business, application software, and ICT platform architectures. *Information and Management, 41*(6), 781-794. doi:10.1016/j.im.2003.06.002

Alghamdi, A. S. (2009). Evaluating defense architecture frameworks for C4I system using analytic hierarchy process. *Journal of Computer Science, 5*(12), 1075-1081. doi:10.3844/jcssp.2009.1075.1081

Almadani, B. (2016). QoS-aware real-time pub/sub middleware for drilling data management in petroleum industry. *Journal of Ambient Intelligence and Humanized Computing*, 287-299.

Alshaer, H. (2015). An overview of network virtualization and cloud network as a service. *International Journal of Network Management*, 1-30.

Armour, F. J., & Kaisler, S. H. (2001). Enterprise architecture: Agile transition and implementation. *IT Professional, 3*(6), 30-37. doi:10.1109/6294.977769

Ashton, K. (2009). That 'Internet of Things' Thing. *RFID Journal*.

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: An experience report. *European Conference on Service-Oriented and Cloud Computing* (pp. 201-215). Springer International Publishing.

Baldwin, C. Y., & Clark, K. B. (2000). *Design Rules: The Power of Modularity Volume 1* (Vol. 1). Cambridge, MA, USA: MIT press.

Bennett, K. B. (2014). Ecological interface design: Military C2 and computer network defense. *IEEE International Conference on Systems, Man and Cybernetics (SMC). 2014-January*, pp. 341-346. IEEE. doi:10.1109/SMC.2014.6973931

Bhanu, S., Babu, A., & Trimurthy, P. (2016). Implementing dynamically evolvable communication with embedded systems through WEB services. *International Journal of Electrical and Computer Engineering*, 381-398.

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 71-79.

Bohara, M., Mishra, M., & Chaudhary, S. (2013). RESTful Web Service integration using Android platform. *4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013.*

Bowen, J. P. (2000). The ethics of safety-critical systems. *Communications of the ACM, 43*(4), 91-97. doi:10.1145/332051.332078

Bowen, J., & Stavridou, V. (1993). Safety-critical systems, formal methods and standards. *Software Engineering Journal, 8*(4), 189-209. doi:10.1049/sej.1993.0025

Britton, C., & Bye, P. (2004). *IT Architectures and Middleware Second Edition Strategies for Building Large, Integrated Systems* (2nd ed.). Boston: Addison-Wesley Professional.

Brooks, F. p. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer, 20*(4), 10-19. doi:10.1109/MC.1987.1663532

Byrd, T. A., & Turner, D. E. (2000). Measuring the Flexibility of Information Technology Infrastructure: Exploratory Analysis of a Construct. *Journal of Management Information Systems, 17*(1), 167-208. doi:10.1080/07421222.2000.11045632

Cemus, K., Cerny, T., Matl, L., & Donahoo, M. (2016). Aspect, rich, and anemic domain models in enterprise information systems. *Lecture Notes in Computer Science*, 445-456.

Chinnici, R., Gudgin, M., Moreau, J.-J., Schlimmer, J., & Weerawarana, S. (2003). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.* W3C.

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer, 27*(8), 44-49. doi:10.1109/2.303623

Deelstra, S., Sinnema, M., & Bosch, J. (2005). Product derivation in software product families: a case study. *Journal of Systems and Software, 74*(2), 173-194. doi:10.1016/j.jss.2003.11.012

Diwan, S. (2016). Toward semantic web using personalization techniques. *International Journal of Applied Engineering Research*, 2451-2453.

EAGLES Evaluation Working Group. (1996). *Evaluation of natural language processing systems. FINAL REPORT.* EAGLES.

Eeratta, R., Shenoy, S., Vijeth, C., & John, N. (2015). Service test automation framework. *Proceedings of 2014 International Conference on Contemporary Computing and Informatics, IC3I 2014*, (pp. 61-66).

Engel, A., & Browning, T. R. (2008). Designing systems for adaptability by means of architecture options. *Systems Engineering, 11*(2), 125-146. doi:10.1002/sys.20090

Fernández-Villamor, J., Iglesias, C., & Garijo, M. (2010). Microservices lightweight service descriptions for REST architectural style. *ICAART 2010 - 2nd International Conference on Agents and Artificial Intelligence*, (pp. 576-579).

Fowler, K. (2004). Mission-critical and safety-critical development. *IEEE Instrumentation and Measurement Magazine, 7*(4), 52-59. doi:10.1109/MIM.2004.1383466

Gallidabino, A., & Pautasso, C. (2016). The Liquid. js Framework for Migrating and Cloning Stateful Web Components across Multiple Devices. *Proceedings of the 25th International Conference Companion on World Wide Web* (pp. 183-186). International World Wide Web Conferences Steering Committee.

Gething, M. J., Williamson, J., Fuller, M., & Ewing, D. (2014). TACTICOS/TACTICOS NC (SABRINA 21)/TACTICOS Compact/ICMS. In *Jane's C4ISR4 & Mission Systems: Maritime* (2nd ed.). IHS.

Ghetas, M., Yong, C., & Sumari, P. (2016). A survey of quality of service in multi-tier web applications. *KSII Transactions on Internet and Information Systems*, 238-256.

Gonzalez, D. (2016). *Developing Microservices with Node.js.* Packt Publishing.

Gorman, J. C., Cooke, N. J., & Winner, J. L. (2006). Measuring team situation awareness in decentralized command and control environments. *Ergonomics, 49*(12-13), 1312-1325. doi:10.1080/00140130600612788

Grolinger, K., Capretz, M., Cunha, A., & Tazi, S. (2014). Integration of business process modeling and Web services: A survey. *Service Oriented Computing and Applications*, 105-128.

Guldentops, E. (2002). Governing information technology through COBIT. In W. Van Grembergen, *Integrity, Internal Control and Security in Information Systems* (pp. 269-309). Springer.

Hakiri, A., Berthou, P., Gokhale, A., Schmidt, D. C., & Thierry, G. (2014). Supporting SIP-based end-to-end data distribution service QoS in WANs. *Journal of Systems and Software*, 100-121.

Halaweh, M. (2012). USING GROUNDED THEORY AS A METHOD FOR SYSTEM REQUIREMENTS ANALYSIS. *Journal of Information Systems and Technolo gy Management, 9*(1), 23-38. doi:10.4301/S1807-17752012000100002

Halili, F., Rufati, E., & Ninka, I. (2013). Styles of service composition - Analysis and comparison methods. *Proceedings - 5th International Conference on Computational Intelligence, Communication Systems, and Networks, CICSyN 2013*, (pp. 302-307).

Haupt, F., Karastoyanova, D., Leymann, F., & Schroth, B. (2014). A model-driven approach for REST compliant services. *Proceedings - 2014 IEEE International Conference on Web Services, ICWS 2014*, (pp. 129-136).

Hill, T., & Westbrook, R. (1997). SWOT analysis: it's time for a product recall. *Long range planning, 30*(1), 46-52. doi:10.1016/S0024-6301(96)00095-7

Hinkin, T. R. (1998). A brief tutorial on the development of measures for use in survey questionnaires. *Organizational research methods, 1*(1), 104-121.

Iacob, M. E., Jonkers, D., Quartel, H., Franken, H., & van den Berg, H. (2012). *Delivering Enterprise Architecture with TOGAF® and ARCHIMATE®.* Enschede: BIZZdesign.

Im, G. P., & Baskerville, R. L. (2005). A Longitudinal Study of Information System Threat Categories: The Enduring Problem of Human Error. *The DATA BASE for Advances in Information Systems, 36*(4), 68-79. doi:10.1145/1104004.1104010

ISO/IEC/IEEE. (2011). *ISO/IEC/IEEE 42010:2011(E): Systems and software engineering -- Architecture description.* ISO/IEC JTC 1/SC 7 Software and systems engineering. Geneva: ISO/IEC. doi:10.1109/IEEESTD.2011.6129467

Josey, A., & Franken, H. (2012). *Archimate® 2.0: An Introduction.* San Francisco, CA, USA: The Open Group. Retrieved from http://www.opengroup.org/subjectareas/enterprise/archimate

Kamrad, B., Schmidt, G. M., & Ulku, S. (2013). Analyzing product architecture under technological change: Modular upgradeability tradeoffs. *IEEE Transactions on Engineering Management, 60*(2), 289-300. doi:10.1109/TEM.2012.2211362

Kang, H., Le, M., & Tao, S. (2016). Container and Microservice Driven Design for Cloud Infrastructure DevOps. *2016 IEEE International Conference on Cloud Engineering (IC2E)* (pp. 202-211). IEEE.

Kovatsch, M., Lanter, M., & Shelby, Z. (2014). Californium: Scalable cloud services for the internet of things with coap. *Internet of Things (IOT), 2014 International Conference on the* (pp. 1-6). IEEE.

Kovatsch, M., Mayer, S., & Ostermaier, B. (2012). Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on* (pp. 751-756). IEEE.

Król, D., & Kitowski, J. (2016). Self-scalable services in service oriented software for cost-effective data farming. *Future Generation Computer Systems*, 1-15.

Lanthaler, M., & Gütl, C. (2013). Hydra: A Vocabulary for Hypermedia-Driven Web APIs. *LDOW*, 996.

Laprie, J. C. (1989). Dependability: A unifying concept for reliable computing and fault tolerance. In T. Anderson, *Dependability of resilient computers* (1st ed., pp. 1-28). Oxford: Blackwell Scientific Publications.

Lasierra, N., Alesanco, A., & Garcia, J. (2014). Designing an architecture for monitoring patients at home: ontologies and web services for clinical and technical management integration. *Biomedical and Health Informatics, IEEE Journal of*, 896-906.

Mari, M., & Eila, N. (2003). The impact of maintainability on component-based software systems. *Conference Proceedings of the EUROMICRO* (pp. 25-32). Belek-Antalya, Turkey: IEEE. doi:10.1109/EURMIC.2003.1231563

Mashal, I., Alsaryrah, O., Chung, T.-Y., Yang, C.-Z., Kuo, W.-H., & Agrawal, D. (2015). Choices for interaction with things on Internet and underlying issues. *Ad Hoc Networks*, 68-90.

Miranda, J., Mäkitalo, N., Garcia-Alonso, J., Berrocal, J., Mikkonen, T., Canal, C., & Murillo, J. (2015). From the Internet of Things to the Internet of People. *IEEE Internet Computing*, 40-47.

Moura, J., & Hutchison, D. (2016). Review and analysis of networking challenges in cloud computing. *Journal of Network and Computer Applications*, 113-129.

Nacer, H., & Aissani, D. (2014). Semantic web services: Standards, applications, challenges and solutions. *Journal of Network and Computer Applications*, 134-151.

Nayebi, F., Desharnais, J. M., & Abran, A. (2012). The state of the art of mobile application usability evaluation. *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering: Vision for a Greener Future, CCECE 2012* (pp. 1-4). Montreal, QC, Canada: IEEE. doi:10.1109/CCECE.2012.6334930

Newman, S. (2015). *Building Microservices.* O'Reilly Media, Inc.

Østerlie, T., & Wang, A. (2006). Establishing maintainability in systems integration: Ambiguity, Negotiations, and infrastructure. *IEEE International Conference on Software Maintenance, ICSM* (pp. 186-195). Philadelphia: IEEE. doi:10.1109/ICSM.2006.27

Paspallis, N., & Papadopoulos, G. A. (2007). An Architecture for Highly Available and Dynamically Upgradeable Web Services. *Advances in Information Systems Development: New Methods and Practice for the Networked Society. 1*, pp. 147-160. Budapest, Hungary: Springer US. doi:10.1007/978-0-387-70761-7_13

Pautasso, C. (2014). RESTful web services: principles, patterns, emerging technologies. In A. Bouguettaya, Q. Z. Sheng, & F. Daniel, *Web services foundations* (pp. 31-51). Springer New York.

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007, 12 01). A design science research methodology for information systems research. *Journal of management information systems, 24*(3), 45-77. doi:10.2753/MIS0742-1222240302

Pérez, I. J., Cabrerizo, F. J., & Herrera-Viedma, E. (2010). A mobile decision support system for dynamic group decision-making problems. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans, 40*(6), 1244-1256. doi:10.1109/TSMCA.2010.2046732

Perry, D., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes, 17*(4), 40-52. doi:10.1145/141874.141884

Petcu, D., Macariu, G., Panica, S., & Crăciun, C. (2013). Portable cloud applications—from theory to practice. *Future Generation Computer Systems*, 1417-1430.

Pilarski, G. (2014). The concept of recommender system supporting command and control system in hierarchical organization. *2014 European Network Intelligence Conference* (pp. 138-141). IEEE. doi:10.1109/ENIC.2014.9

Pillutla, S. (2014). Issues in porting a LAN-based total enterprise simulation game to a Web-based environment. *Developments in Business Simulation and Experiential Learning*, 35.

Pintus, A., Carboni, D., & Piras, A. (2012). Paraimpu: a platform for a social web of things. *Proceedings of the 21st international conference companion on World Wide Web* (pp. 401-404). ACM.

Project Management Institute. (2004). *A guide to the project management body of knowledge: PMBOK guide* (3rd ed.). Project Management Institute.

Rohprimardho. (2015). *Measuring The Impact of Docker on Network I/O Performance.* Amsterdam: University of Amsterdam.

Rouhani, B. D., Mahrin, M. N., Nikpay, F., & Rouhani, B. D. (2014). Current issues on enterprise architecture implementation methodology. *2014 World Conference on Information Systems and Technologies, WorldCIST 2014. 276* , pp. 239-246. Kuala Lumpur, Malaysia: Springer Verlag. doi:10.1007/978-3-319-05948-8_23

Ryan, C., & Garland, R. (1999). The use of a specific non-response option on Likert-type scales. *Tourism Management, 20*(1), 107-113.

Santos, C., Junior, J., Soares, A., Carneiro, N., Araujo, T., Miranda, B., & Serique, B. (2015). Service oriented architecture for data visualization in smart devices. *Proceedings of the International Conference on Information Visualisation, 2015-September*, (pp. 561-567).

Scott-Hayward, S., Natarajan, S., & Sezer, S. (2016). survey of security in software defined networks. *IEEE Communications Surveys and Tutorials*, 623-654.

Sergeevich, K., Ovseevna, A., & Petrovich, S. (2015). a Web-application for real-time big data visualization of complex physical experiments. *2015 International Siberian Conference on Control and Communications, SIBCON 2015* .

Shaukat, U., Ahmed, E., Anwar, Z., & Xia, F. (2016). Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 18-40.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences, 280*, 218-238.

Spaccapietra, S., Al-Jadir, L., & Yu, S. (2005). Somebody, sometime, somewhere, something. *Proceedings of the International Workshop on Ubiquitous Data Management , UDM 2005* (pp. 6-13). Tokyo, Japan: IEEE. doi:10.1109/UDM.2005.20

Thales Group. (2015, June). *For a safer world.* Retrieved 02 19, 2016, from Thalesgroup.com: https://www.thalesgroup.com/sites/default/files/asset/document/13513_thales_brochure_gb.pdf

Thales Nederland. (2008). *Tacticos Combat Management System: Continous improvement through constant innovation.* Retrieved 02 22, 2016, from thales7seas.com: http://www.thales7seas.com/html_2014/product77.html

Thales Nederland. (2013). *Tacticos: the best just got better.* Retrieved 02 22, 2016, from thales7seas.com: http://www.thales7seas.com/html_2014/product77.html

Thomas, L. (1986). A survey of maintenance and replacement models for maintainability and reliability of multi-item systems. *Reliability Engineering, 16*(4), 297-309. doi:10.1016/0143-8174(86)90099-5

Thönes, J. (2015). Microservices. *IEEE Software*, 112-115.

Tolk, A. (2012). *Engineering Principles of Combat Modeling and Distributed Simulation.* (A. Tolk, Ed.) Hoboken, NJ, USA: John Wiley & Sons, Inc. doi:10.1002/9781118180310.ch3

Traore, B., Kamsu-Foguem, B., & Tangara, F. (2016). Integrating MDA and SOA for improving telemedicine services. *Telematics and Informatics*, 733-741.

van den Berg, M., Tang, A., & Farenhorst, R. (2009). A constraint-oriented approach to software architecture design. *Quality Software, 2009. QSIC'09. 9th International Conference on* (pp. 396-405). IEEE. doi:10.1109/QSIC.2009.59

Vasista, T., & Alsudairi, M. (2013). Service-Oriented Architecture (SOA) and semantic web services for web portal integration. *Advances in Intelligent Systems and Computing*, 253-261.

Vega-Gorgojo, G., Gómez-Sánchez, E., Bote-Lorenzo, M., & Asensio-Pérez, J. (2012). RESTifying a legacy semantic search system: Experience and lessons learned. *Journal of Universal Computer Science, 18*(2), 286-311.

Velasco, M., While, D., & Raju, P. (2013). Adaptation of Web services using policies. *2013 8th International Conference for Internet Technology and Secured Transactions, ICITST 2013*, (pp. 338-343).

Wagner, S. (2013). *Software product quality control.* Stuttgart, Germany: Springer. doi:10.1007/978-3-642-38571-1

Wang, L., & Canedo, A. (2014). Offloading industrial human-machine interaction tasks to mobile devices and the cloud. *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014* (pp. 1-4). Barcelona, Spain: IEEE. doi:10.1109/ETFA.2014.7005249

Zhang, W., Hansen, K., & Ingstrup, M. (2014). A hybrid approach to self-management in a pervasive service middleware. *Knowledge-Based Systems*, 143-161.

Zhou, J., Leppanen, T., Harjula, E., Ylianttila, M., Ojala, T., Yu, C., & Yang, L. T. (2013). Cloudthings: A common architecture for integrating the internet of things with cloud computing. *Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on* (pp. 651-657). IEEE.

Zhu, J., Chan, D. S., Prabhu, M. S., Natarajan, P., Hu, H., & Bonomi, F. (2013). Improving web sites performance using edge servers in fog computing architecture. *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on* (pp. 320-323). IEEE.

Zur Muehlen, M., & Ho, D.-Y. (2005). Risk management in the BPM lifecycle. *BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS. 3812 LNCS*, pp. 454-466. Howe School of Technology Management, Stevens Institute of Technology, Castle Point on the Hudson, Hoboken, NJ 07030, United States: Springer Verlag. doi:10.1007/11678564_42

Zuzak, I., Budiseli, I., & Delac, G. (2011). A finite-state machine approach for modeling and analyzing Restful systems., (pp. 353-390).

Zuzak, I., Budiselic, I., & Delac, G. (2011). Formal modeling of RESTful systems using finite-state machines., (pp. 346-360).

List Of Figures And Tables

The image on the front cover of this thesis is an adaptation of Figure 19: "Redundancy measures using multiple web applications servers" on page 37.

Figures

FIGURE 1: THE RELATIONS BETWEEN THE RESEARCH QUESTIONS AND THE STEPS IN THE DSRM PROCESS 6
FIGURE 2: (ANONYMISED) TABLE OF THE INTERVIEWED EMPLOYEES AND THEIR COVERAGE OF THE DEPARTMENTS 10
FIGURE 3: ADAPTED SWOT-ANALYSIS BASED ON 'CURRENT ARCHITECTURE VS. NEW ARCHITECTURE' INSTEAD OF 'INTERNAL VS. EXTERNAL' 11
FIGURE 4: A SELECTION OF CREWMEMBER ROLES WITHIN A NAVAL VESSEL 17
FIGURE 5: A SELECTION OF FUNCTIONS THAT CAN BE PERFORMED WITH THE TACTICOS SYSTEM 17
FIGURE 6: GLOBAL OVERVIEW OF A SELECTION OF THE (SUB)SYSTEMS. 18
FIGURE 7: CONTENTS OF AN ENVIRONMENTAL CONDITIONING CABINET SERVER RACK 19
FIGURE 8: OVERVIEW OF A MULTIFUNCTION OPERATOR CONSOLE 20
FIGURE 9: (PHYSICAL) NETWORK LAYOUT FOR REDUNDANCY. 21
FIGURE 10: INTERCONNECTIONS BETWEEN (SUB)-SYSTEMS AND VLAN'S. 22
FIGURE 11: THE OPENSPLICE DATA DISTRIBUTION SERVICE 23
FIGURE 12: APPLICATION INTERACTION BETWEEN 3D AIR SURVEILLANCE RADAR AND THE MULTIFUNCTION OPERATOR CONSOLE. . 24
FIGURE 13: APPLICATION FLOW FOR NON-REAL-TIME (STANDARD LATENCY) VIDEO. 25
FIGURE 14: APPLICATION FLOW FOR REAL-TIME (LOW-LATENCY) VIDEO 26
FIGURE 15: APPLICATION FLOW FOR VIDEO REPLAYS 27
FIGURE 16: THE OVERVIEW OF THE APPLICATIONS AND HARDWARE SUPPORTING THE THREE VIDEO FLOWS 28
FIGURE 17: THE ARCHITECTURE INVOLVED IN TRAINING SIMULATIONS 29
FIGURE 18: CONTAINERIZED MICROSERVICES ON A WEB APPLICATIONS SERVER 36
FIGURE 19: REDUNDANCY MEASURES USING MULTIPLE WEB APPLICATIONS SERVERS 37
FIGURE 20: NEW WEB TECHNOLOGY FOR THE 3D SEARCH RADAR USE CASE 39
FIGURE 21: NEW WEB TECHNOLOGY FOR THE VIDEO SERVER USE CASE 42
FIGURE 22: ADDITIONAL WEB APPLICATIONS FOR FUTURE FUNCTIONALITY IN THE SIMULATION ARCHITECTURE 44
FIGURE 23: END STATE OF THE SIMULATION ARCHITECTURE 45
FIGURE 24: SIMULATION ARCHITECTURE IN THE 3D SEARCH RADAR USE CASE 47
FIGURE 25: WEB APPLICATIONS SERVER - SINGLE NODE 49
FIGURE 26: CENTRAL WEB APPLICATIONS SERVERS - REDUNDANT NODES 50
FIGURE 27: THE DUAL WEB APPLICATION SERVERS WITHIN THE CONTEXT OF THE PHYSICAL NETWORK 51
FIGURE 28: ARCHITECTURE OF DEMONSTRATOR 1 54
FIGURE 29: SCREENSHOT OF DEMONSTRATOR 1 55
FIGURE 30: ARCHITECTURE OF DEMONSTRATOR 2 56
FIGURE 31: SCREENSHOT OF THE CONTROL PAGE IN DEMONSTRATOR 2 57
FIGURE 32: SCREENSHOT OF THE MASTER PAGE IN DEMONSTRATOR 2 57
FIGURE 33: ARCHITECTURE OF DEMONSTRATOR 3 58
FIGURE 34: SCREENSHOT OF DEMONSTRATOR 3 59
FIGURE 35: ARCHITECTURE OF DEMONSTRATOR 4 60
FIGURE 36: SCREENSHOT OF THE DEMONSTRATOR 4 BROWSER WINDOW WITH A LIVE SERVER 61
FIGURE 37: SCREENSHOT OF THE DEMONSTRATOR 4 BROWSER WINDOW WITH NO SERVER TO CONNECT TO 61
FIGURE 38: ARCHITECTURE OF DEMONSTRATOR 5 62
FIGURE 39: SCREENSHOT OF DEMONSTRATOR 5 63

FIGURE 40: ARCHITECTURE OF DEMONSTRATOR 6 64
FIGURE 41: NON-SECURE CONNECTION TO SECURE SENSORSIM-A 65
FIGURE 42: SECURE CONNECTION TO SECURE SENSORSIM-A 65
FIGURE 43: DETAILS OF SECURE CONNECTION TO SENSORSIM-A 66
FIGURE 44: NON-SECURE CONNECTION TO COMPROMISED SENSORSIM-B 66
FIGURE 45: SECURE CONNECTION TO COMPROMISED SENSORSIM-B 67
FIGURE 46: SECURE CONNECTION TO SENSORSIM-C THROUGH HTTPS-ENDPOINT 67

Tables

TABLE 1: RESEARCH QUESTIONS 4
TABLE 2: RELATIONS BETWEEN THESIS CHAPTERS, RESEARCH QUESTIONS, AND THE DSRM 7
TABLE 3: REQUIREMENTS FOR THE NEW ARCHITECTURE 15
TABLE 4: SEARCH TERMS USED FOR THE LITERATURE REVIEW 31
TABLE 5: RESULTS OF THE ARCHITECTURE EVALUATION QUESTIONNAIRE 53
TABLE 6: SEARCH TERMS USED IN THE SCOPUS DATABASE 81
TABLE 7: SEARCH TERMS USED IN THE GOOGLE SCHOLAR SEARCH ENGINE 82
TABLE 8: THE SOURCES IN WHICH THE STYLES & PATTERNS WERE FOUND IN THE LITERATURE REVIEW. 84

Index

An explanation or important occurrence of the following terms can be found at the indicated page numbers.

ADL, 16, *See* Architecture description language
Archimate, 16
Implementation and migration extension, 16
Motivation extension, 16
Architectural description, 16
Architectural styles, 31
Architecture description language, 16
Architectures, 16
Atom Electron, 40
C2 systems, 2, *See* Command & control systems
Chromium, 40
client-server architecture, 34
Command & control systems, 2
Community cloud, 33
Decentralized command & control systems, 2, *See* Command & control systems
Design science research methodology, 5, 8
Docker, 33
DSRM, 5, *See* Design science research methodology
EAF, 31, *See* Enterprise architecture framework
Enterprise architecture framework, 31
Gluster, 48
Google Chrome browser, 40
Google V8 Javascript engine, 40
Human factors, 2
Hybrid cloud, 33
Infrastructure as a Service (IaaS), 33
ISO/IEC 25010 standard, 3, 4
layered systems, 34
Microservices, 32
Mission-critical system, 2
NFR. *See* Non-functional requirements
NGINX, 54
NodeJS, 40
Platform as a Service (PaaS), 33
Private cloud, 33
Public cloud, 33
Representational State Transfer, 32
RESTful web services, 33
Safety-critical system, 2
Service Oriented Architecture, 32
Services, 32
Situational awareness, 2
Software as a Service (SaaS), 33
Transition plan. *See* Migration plan
Web services, 33

Appendix A: Search terms used in the literature review

In this appendix the exact search terms as used for the literature search in section 4.1 are listed. The search terms for the Scopus database are shown as outputted by the database. The search terms for the Google Scholar search engine are shown with the selected search-settings enclosed in brackets.

Search term identifier	Search term
Scopus 01	TITLE-ABS-KEY (architectural styles web development) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014) OR LIMIT-TO (PUBYEAR , 2013) OR LIMIT-TO (PUBYEAR , 2012) OR LIMIT-TO (PUBYEAR , 2011) OR LIMIT-TO (PUBYEAR , 2010))
Scopus 02	(TITLE-ABS-KEY (lan-based) AND PUBYEAR > 2009) AND (architecture) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI"))
Scopus 03	TITLE-ABS-KEY (web architectures) AND PUBYEAR > 2009 AND (LIMIT-TO (DOCTYPE , "re")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014))
Scopus 04	TITLE-ABS-KEY (network architectures) AND PUBYEAR > 2009 AND (LIMIT-TO (DOCTYPE , "re")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016))
Scopus 05	TITLE-ABS-KEY (web server architectures) AND PUBYEAR > 2009 AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016))
Scopus 06	(TITLE-ABS-KEY (software architectures)) AND ((web)) AND (patterns) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO (DOCTYPE , "re")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016))
Scopus 07	(TITLE-ABS-KEY (systems of systems)) AND ((architectures)) AND (mission critical) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO (DOCTYPE , "re")) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI")) AND (LIMIT-TO (PUBYEAR , 2016))

Table 6: Search terms used in the Scopus database

Search term identifier	Search term
Scholar 01	architectural styles web development (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 02	LAN-based architecture (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 03	web architectures (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 04	network architectures (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 05	web server architectures (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 06	software architectures web patterns (period: Sinds 2012)(Results used: 20)(excl. patents)
Scholar 07	systems of systems architectures mission critical (period: Sinds 2012)(Results used: 20)(excl. patents)

Table 7: Search terms used in the Google Scholar search engine

Appendix B: Architectural styles found in the literature search

The architectural styles found in the (small-scale) systematic literature search can be seen in Table 8. In the sources in which the different concepts were found are related to the different concepts. Sorting firstly by number of sources in the category, and secondly by alphabet.

Style/Pattern	Found in reviewed sources:
Representational State Transfer	(Zuzak, Budiselic, & Delac, Formal modeling of RESTful systems using finite-state machines, 2011) (Zuzak, Budiseli, & Delac, A finite-state machine approach for modeling and analyzing Restful systems, 2011) (Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012) (Halili, Rufati, & Ninka, 2013) (Bohara, Mishra, & Chaudhary, 2013) (Zhang, Hansen, & Ingstrup, 2014) (Haupt, Karastoyanova, Leymann, & Schroth, 2014) (Sergeevich, Ovseevna, & Petrovich, 2015) (Santos, et al., 2015) (Lasierra, Alesanco, & Garcia, 2014) (Sheng, et al., 2014)
Service-Oriented Architectures	(Vasista & Alsudairi, 2013) (Velasco, While, & Raju, 2013) (Halili, Rufati, & Ninka, 2013) (Zhang, Hansen, & Ingstrup, 2014) (Eeratta, Shenoy, Vijeth, & John, 2015) (Santos, et al., 2015) (Nacer & Aissani, 2014) (Grolinger, Capretz, Cunha, & Tazi, 2014) (Traore, Kamsu-Foguem, & Tangara, 2016) (Sheng, et al., 2014) (Pautasso, 2014)
Web Services	(Velasco, While, & Raju, 2013) (Santos, et al., 2015) (Nacer & Aissani, 2014) (Grolinger, Capretz, Cunha, & Tazi, 2014) (Traore, Kamsu-Foguem, & Tangara, 2016) (Bhanu, Babu, & Trimurthy, 2016) (Lasierra, Alesanco, & Garcia, 2014) (Sheng, et al., 2014) (AbuJarour & Awad, 2014)
Cloud Computing	(Alshaer, 2015) (Moura & Hutchison, 2016) (Shaukat, Ahmed, Anwar, & Xia, 2016) (Petcu, Macariu, Panica, & Crăciun, 2013) (Zhou, et al., 2013)
Layered Style	(Zhang, Hansen, & Ingstrup, 2014) (Haupt, Karastoyanova, Leymann, & Schroth, 2014) (Cemus, Cerny, Matl, & Donahoo, 2016) (Petcu, Macariu, Panica, & Crăciun, 2013) (Pillutla, 2014)
RESTful Web Services	(Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012) (Bohara, Mishra, & Chaudhary, 2013) (Grolinger, Capretz, Cunha, & Tazi, 2014) (Sheng, et al., 2014) (Pautasso, 2014)
Semantic Web Services	(Nacer & Aissani, 2014) (Grolinger, Capretz, Cunha, & Tazi, 2014) (Lanthaler & Gütl, 2013) (Vasista & Alsudairi, 2013)
Client-server	(Vega-Gorgojo, Gómez-Sánchez, Bote-Lorenzo, & Asensio-Pérez, 2012) (Santos, et al., 2015) (Pillutla, 2014)
Internet of Things	(Kovatsch, Mayer, & Ostermaier, 2012) (Zhou, et al., 2013) (Kovatsch, Lanter, & Shelby, 2014)
Web of Things	(Mashal, et al., 2015) (Pintus, Carboni, & Piras, 2012) (Kovatsch, Mayer, & Ostermaier, 2012) (Kovatsch, Lanter, & Shelby, 2014)
Cloudlet	(Shaukat, Ahmed, Anwar, & Xia, 2016) (Petcu, Macariu, Panica, & Crăciun, 2013)
Data Distribution Service	(Almadani, 2016) (Hakiri, Berthou, Gokhale, Schmidt, & Thierry, 2014)
Mobile Cloud Computing	(Moura & Hutchison, 2016) (Shaukat, Ahmed, Anwar, & Xia, 2016)
Publish/subscribe Style	(Zhang, Hansen, & Ingstrup, 2014) (Almadani, 2016)
Social Web of Things	(Mashal, et al., 2015) (Pintus, Carboni, & Piras, 2012)
Software Defined Networking	(Moura & Hutchison, 2016) (Scott-Hayward, Natarajan, & Sezer, 2016)
Centralized data warehouse	(Sergeevich, Ovseevna, & Petrovich, 2015)
Community cloud	(Alshaer, 2015)
EntityControl-Boundary	(Santos, et al., 2015)

pattern	
Event-based style	(Zhang, Hansen, & Ingstrup, 2014)
Five layer IoT architecture	(Mashal, et al., 2015)
Fog Computing	(Zhu, et al., 2013)
Hexagonal architecture	(Santos, et al., 2015)
Hybrid cloud computing	(Alshaer, 2015)
Internet of People	(Miranda, et al., 2015)
Liquid software	(Gallidabino & Pautasso, 2016)
Microservices	(Fernández-Villamor, Iglesias, & Garijo, 2010)
Model/View Architecture	(Sergeevich, Ovseevna, & Petrovich, 2015)
Model/View/Controller Pattern	(Santos, et al., 2015)
Multi-tier web application architecture	(Ghetas, Yong, & Sumari, 2016)
Peer-to-Peer style	(Zhang, Hansen, & Ingstrup, 2014)
Private cloud	(Alshaer, 2015)
Repository style	(Zhang, Hansen, & Ingstrup, 2014)
Self-scalable services	(Król & Kitowski, 2016)
Semantic Web	(Diwan, 2016)
Service-component architecture	(Zhang, Hansen, & Ingstrup, 2014)
Social Internet of Things	(Mashal, et al., 2015)
Thick-client	(Sergeevich, Ovseevna, & Petrovich, 2015)
Thin server architecture	(Kovatsch, Mayer, & Ostermaier, 2012)
Three layer IoT architecture	(Mashal, et al., 2015)
Virtual networking	(Moura & Hutchison, 2016)
Virtual Web Services	(Nacer & Aissani, 2014)

Table 8: The sources in which the styles & patterns were found in the literature review.

Appendix C: Architecture evaluation questionnaire

Architecture evaluation form

Name: Click here to enter text.

Function: Click here to enter text.

Department: Click here to enter text.

For each of the requirements stated, mark either No Opinion, Strongly disagree, Disagree, 'Neither agree nor disagree', Agree, or Agree strongly, to best represent your opinion in regard to whether the new architecture would be an improvement over the current architecture of the TACTICOS architecture.

The new architecture would better support REQ-XX.	No opinion	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Agree strongly
REQ-01: The system should be modular						
REQ-02: Changes in functionality should not require the entire system to be updated.						
REQ-03: Applications should be decoupled from each other, so tasks can be done when there are resources available, and failures don't cascade to other applications.						
REQ-04: There should be less dependency on a continuous connection between client and server						
REQ-05: The technology used for the GUI should be modern and easy to make changes to.						
REQ-06: Prototyping should be rapid and done often						
REQ-07: Processing should be done more on the server-side instead of on the client-side.						
REQ-08: There should be more of a focus on security within the network.						
REQ-09: Data should be protected from unwarranted access within the network.						
REQ-10: Expansion to mobile devices should be supported						
REQ-11: The system needs to be maintained for 15-20 years						
REQ-12: Applications/systems should be designed for testability.						
REQ-13: Use technology that is mainstream/common, so 3rd party components and personnel are easier to be found.						
REQ-14: The deployment of a new architecture should be done incrementally and co-exist with legacy technology						